ESD-TR-74-307

DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE

R. K. Overton, et al
AMS, Inc.
401 N. Harvard Avenue
Claremont, CA 91711

September 1974

Approved for public release;
distribution unlimited.

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
HANSCOM AFB, MA 01731

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

This technical report has been reviewed and is approved for publication.

George E. Reynolds	Sylvia R. Mayer
GEORGE E. REYNOLDS Task Scientist Graphics Techniques for Software Maintenance	SYLVIA R. MAYER, Ph.D. Project Scientist, Project 2801 System Design Methodology

FOR THE COMMANDER

Robert W O'Keefe

ROBERT W. O'KEEFE, Colonel, USAF
Director, Information Systems
Technology Applications Office
Deputy for Command & Management Systems

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-74-307	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. K. Overton, et al		8. CONTRACT OR GRANT NUMBER(s) F19628-74-C-0061
9. PERFORMING ORGANIZATION NAME AND ADDRESS AMS, Incorporated 401 N. Harvard Avenue Claremont, CA 91711		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element 62702F Project No. 2801 Task Area 19 Work Unit No. 003
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Command and Management Systems Hq Electronic Systems Division (AFSC) Hanscom AFB, MA 01731		12. REPORT DATE September 1974 13. NUMBER OF PAGES 263
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

System Software Maintenance
Conceptual Groupings
Sensory Integration
Computer Aids

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Data were collected on two aspects of maintenance programming (which, according to published estimates, costs the U.S. approximately five billion dollars a year). Aspects were (1) arrangement and sources of information at graphics consoles, and (2) the value of "conceptual groupings" to maintenance programmers using FORTRAN and PL/1.

Item 20 (continued)

At consoles, there is a need for a better matching of problem-solving facilities with the level of abstraction or detail at which the programmer happens to be working. Scattered sources of needed information were a handicap, as was distraction and other factors. It would be possible to develop some techniques for reducing these handicaps.

Pilot programs were developed to automate display of "conceptual groupings." In at least some cases, such programs decidedly improve maintenance efficiency. Further development and wider usage of such programs is warranted.

DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE

DCASM Final Report

Table of Contents

Table of Contents (continued)

Table of Contents (continued)

DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE

DCASM Final Report

List of Figures and Tables

Page No.

Figure

Table

1. INTRODUCTION: NEEDS AND AREAS

1.1 Needs for Research

Within the broad field of computer programming, one need is growing particularly; and awareness of that need is also growing. That need is the facilitation of maintenance programming, by programmers who did not originally develop a software system, on software which others have written.

Several factors cause the need:

(1) As programming becomes less of a pioneering work, there simply are more programs which one may modify rather than develop "from scratch." Also, as telecommunications are improved and as computers become larger, a tendency is reported (Hill, 1973) for specialized software packages to be shared (and modified) by a number of users.

(2) Personnel turnover has traditionally been high in programming. While it may have declined in recent years, it is still high in relation to other professions: One study (Simmons, 1972) estimates that twenty percent of U.S. programmers move from one employer to another each year.

(3) There is a tendency for software systems to become more ambitious; and larger systems, according to Simmons (1972) amplify the need.

High costs have resulted from these and other factors. For the U.S. alone, according to one student of the problem (Boehm, 1973), annual " . . . software costs . . . are probably over $10 billion . . .".

Maintenance programming is a major factor in these costs. The U.S.A.F. Systems Command, Electronic Systems Division, in sponsoring previous research in this area, estimated that the maintenance of a large computer program often costs more, over the life of the program, than was paid for the original development of the program. And a recent managers' survey (Rhodes, 1973) showed that the typical cost of developing "a big system" was greatly surpassed by the costs of installing and testing it, and of making enhancements in it. If published estimates can be believed, this all implies that maintenance costs the U.S. more than $5 billion per year.

In spite of the involvement of billions of dollars, there has been a remarkable lack of systematic research into the conditions which help and hinder people doing maintenance programming. Electronic Systems Division, recognizing this lack, has sponsored relevant research (e.g., Overton et al., 1973). This helped correct what was earlier described as " . . . an applied scientific lag in the study of . . . programming . . . . a widening and critical lag which threatens . . . great waste that inevitably accompanies the absence of systematic and established methods and findings, and their substitution by anecdotal opinion, vested interests, and provencialism" (Sackman et al., 1968).

The previous research identified areas which promised particularly great potential for development--from which helpful tools and practical advice could come. The present project represents further work in two of those areas. They are called "Graphic Terminal Arrangements" and "Automated Conceptual Groupings."

1.2 Areas of Research

The nature of the work reported here is transitional, between research and development. One area (Conceptual Groupings) includes the development of prototype programs which could be converted into valuable tools for maintenance programmers. The other area (Terminal Arrangements) includes research into the conditions which facilitate problem-solving in programming.

1.2.1 Graphic Terminal Arrangements

Through his eyes and other senses, the programmer at a terminal takes in a variety of information. The integration, or lack thereof, of this information, can make the programmer's job relatively easy, or very inconvenient. Thus, this area of research concerned the integration of the information used by the person at the terminal.

There are complex interactions between the physical arrangements, the programming discipline in which a person is working, and the style and structure of the program. This research sought to clarify the effects of some of the variables, so that effective arrangements can be devised for future work situations.

In addition to studying the basic variables, the project also pointed out some specific, smaller features of terminal arrangements which are both helpful and harmful.

1.2.2 Automated Conceptual Groupings

When one programmer is talking to another, explaining a program to him, much of his explanation is cast in interesting units: not individual statements of code, and not in the terms of the formal documentation of the program; but in small sets of statements, each set of which can be treated as a unit. These sets have been called "Conceptual Groups." This project included work on programs which would find "Conceptual Groupings" automatically.

Two pilot versions of "Grouping Programs" were written. Research showed that such programs could help make at least certain classes of applications programs significantly easier to maintain.

The logic of the programs could also provide the designs by which compilers could be modified, so that the compilers could, by helping the programmers, reduce the cost of maintaining programs.

2. STUDIES IN GRAPHIC TERMINAL ARRANGEMENTS

2.1 General Themes

Although they are not often repeated explicitly, two general themes run through the studies in terminal arrangements.

First, there are limitations to what people's senses can perceive; that, both literally and figuratively, eyes have to blink now and then; that some things are hard to see and follow; and that people may get tired of trying.

A second theme relates to more intellectual (as opposed to sensory) functions. Often, the difficulty of a problem is a function of the way it is presented. And, in a sense, a terminal is a device for presenting a problem, or a portion of a problem, to a maintenance programmer.

The first (or more sensory) theme appears strongly in the study of the literature (described below) relevant to this area, and in the direct observations of the problems of maintenance programmers at terminals.

The second (or more intellectual) theme develops into experimentation. But before the experiments could be executed, a methodology had to be developed—problems had to be presented in different ways, so that the best feasible way could be found. The experimentation, then, consisted of programmers actually working on programming problems which differed significantly and systematically in complexity and mode of presentation.

The two themes are summed up in the phrase, "sensory integration." The general goal is to present information relevant to a maintenance programming problem in the way in which the information is easiest for people to (1) perceive, and (2) integrate into a solution to the problem.

2.2 Research Methodology

The following activities were included as methods of study of Terminal Arrangements:

(1) Relevant literature was extensively reviewed. (This review was combined with that for the Conceptual Groupings studies.) Basic findings are incorporated in the body of this report, but additional quotations appear in Appendix A.

(2) Concepts--especially those necessary to the making of specific definitions about working conditions and performance--were developed.

(3) Many possible experimental variables were refined into a tractable number, and an experimental design was developed.

(4) Observations--including interviews--and experiments--including videotaping--were conducted.

(5) Results of experiments were predicted theoretically, and actual results were compared with the theory.

These activities are discussed, in turn, below.

2.2.1 Literature Review

A wide spectrum of sources was represented in the literature which was reviewed. These sources may be roughly divided into four classes:

(1) Professional computing journals. Examples include the IEEE and ACM journals, and others of lower circulation.

(2) Other professional journals. These, such as Perceptual and Motor Skills, may contain relevant facts while not being oriented toward computers or programming.

(3) Popular or semi-professional magazines. A prime example is Datamation. (The authors observe that many popular magazines contain very little "firm" information; but they may be of value in reporting current opinions and interests.)

(4) Other publications. Included are books, dissertations, and symposium proceedings.

More than 200 articles, etc., were read during the review. (Of course a larger number were merely scanned briefly or skimmed, and rejected as being not worth perusing.) These were converted into a file of relevant findings, etc. Where appropriate, these were incorporated into the text of this Final Report.

Several of the sources contained "quotable quotes" which were interesting, but which would not fit appropriately into the body of the report. Some of these quotations were copied and included in Appendix A.

2.2.2 Concept Development

One of the problems in trying to study maintenance programming scientifically, as previously observed (Overton et al., 1973, p. 84)

> " . . . is that the really meaningful elements of the work are difficult to define. That is, the programmer is not like a factory worker who is installing components that can be counted . . . . He is more like an artisan who is figuring out how to make or repair an artifact. (The word 'artifact' is used in its archeological sense, as 'a product of human workmanship.') His 'figuring out' is hard to translate into anything like a quantitative statement of the ingredients in his problem-solving.
>
> "With a time-sharing system, however, the programmer enters an environment in which one element of his work is theoretically subject to regular and continual measurement. That element is <u>information</u>. The programmer is calling for information to be displayed on his terminal. He is (mentally) processing that information, and he is throughputting the results of the processed information by making new inputs at the terminal."

The final assumption is deceptively simple: The more information the programmer throughputs, and the more accurately he does it, the better he is working.

Unfortunately, one must develop more specific and observable concepts before one can use this assumption as a guide to the measurement of programmer productivity. In this connection, a new body of literature becomes relevant. It is that on "operator workload": the "load" a man can "carry" when he must respond to inputs from the electronic portions of a man-machine system of which he is a part.

A recent review of this literature (Jahns, 1973) emphasizes the diverse origins of the one concept of the (sensory and intellectual) "workload." The concept is said to derive from four different classes of researchers and theorists, having somewhat different meanings for each of them. The four are:

(1) Time-and-motion analysts. Using a "scenario" as a guide, lengthy predictions of the exact tasks with which an operator will be faced. They also predict the time it will take him to do each action. These predictions are based on normative data from previous studies. The specialists then calculate the points in time when he simply will not have enough time to do what he is supposed to do.

This technique is said to " . . . provide an adequate tool for making general, broad predictions regarding the operator's ability to handle a given set of tasks."

The barrier to applying this technique in maintenance programming is that one does not usually know in advance what the "given set of tasks" will be. Also, there is a lack of normative data on the time required to do various tasks.

(2) "Channel Capacity" theorists. It is assumed that the operator works within a "channel capacity" which is fixed and limited. "Workload" capacity is the ability to accomplish additional tasks, whether expected or unexpected.

This approach is said to be valuable when applied to a small number of individual functions; but it may distort the picture if it is applied to "complex interactions" such as those found in "higher-order man-machine systems."

This approach is applicable to some aspects of maintenance programming, such as the simple scanning of files for points of interest.

(3) "Activation" theorists. Performance is said to be a function of the activation level, or physiological state, of the operator.

"The results obtained so far are promising but fraught with problems in measurement techniques, data reduction and interpretation."

In the opinion of the authors, this approach has no place, at present, in the study of maintenance programming.

(4) Non-theoreticians. It is assumed that anything which will "simplify" the work situation will help the operator. Also, any situational change which improves performance is assumed to have reduced the sensory/intellectual workload.

Now, here are two concepts which were actually used in this project, and their relationship to the concepts from the literature as reviewed by Jahns (1973).

(1) Information sources. The screen of a terminal is an information source, as is a listing of the program. As will be seen, there are other sources of information which are quite important to the maintenance programmer.

Within the time-and-motion tradition, it is assumed that a programmer can work better when he has to deal with a smaller number of sources at one time, and when each is as convenient to him as possible.

(2) Throughput rate. When a maintenance programmer looks at a new item on a display, he has to decide, at a minimum whether or not it is relevant to what he is doing at the moment. There is a maximum rate at which he can make such decisions, and this rate will therefore tend to set a pace for his work.

This concept is compatible with that of "channel capacity." The authors prefer to express the throughput rate, wherever it is possible to do so, in terms of <u>bits per second</u>.

Another concept, unrelated to the "workload" literature, relates to people's ability at "Keeping Track of Several Things at Once" (Yntema, 1963). There is a limit to this ability too; and it becomes very relevant to a maintenance programmer when he needs simultaneously to watch more than one source of information.

A "proper" experimental design is based upon variables --controlled, independent, and dependent variables--and not upon general concepts. But the above concepts were used in evaluating the variables described below.

2.2.3 Experimental Variables and Design

People constantly adapt to, and overcome, small imperfections in their equipment. On the other hand, people also respond to, and are distracted by, many "non-signals" which computers ignore. These characteristics of people contributed to making the experimental design (potentially) extremely complex. Maintenance programming is simply very complex behavior, and there is no realistic way around this fact.

One could easily list more than a dozen variables (such as those cited below) which influence maintenance programming. A necessary task was the refinement of these variables into those which were most basic.

Then an experimental design (described below) was developed in terms of the basic variables.

Finally, to keep the results from being unrealistically narrow, experimentation plus observations were planned. (This combination is also described below.)

2.2.3.1 Variable Refinement. Preliminary observations, coordinated with the literature review, led to a formidable list of independent variables upon whose effects the productivity and success of maintenance programmers might in part depend. These include:

(1) Language. The pattern of errors which programmers make in working with different languages has been found to vary from language to language (Youngs, 1969). FORTRAN and PL/1 probably differ from each other, and from other languages, in specific aspects of maintainability.

(2) Terminal. A graphics terminal is obviously not equivalent, in terms of the information which a person can get from it, and the way he can get that information, to older-style print-outs and "hard copy." The terminal itself may affect maintenance behavior.

(3) Other information sources. Accessibility of other data, such as printed documentation, may be necessary to complement the information received from terminals and program listings (if any). Ideally, experimentation should cover these other information sources.

(4) Program application. Payroll programs have been written in both FORTRAN and COBOL, as well as in PL/1 and other languages. Within the framework of one language, the application area may affect maintainability. And there obviously are different types of applications, including (1) business, (2) scientific, (3) statistical, (4) simulation, and (5) others.

(5) Size. It seems reasonable to suppose that a big program should be harder to maintain than a small one, but there are no "hard" data on the relationship between maintainability and size. Conceivably, this experimentation might seek to generate such data.

(6) Style. Under at least some conditions, programmers prefer to work on programs which are written in a highly modular, hierarchical style (Overton et al., 1973). Hence aspects of style, such as degree of modularization, are candidates for experimental variables.

(7) Test difficulty. A commonly-expressed opinion is that, especially with larger software systems, the ease of maintenance work depends in part on the availability of test data, and the convenience of test conditions.

(8) Complexity. One program may differ considerably from another in terms of logical consequences and complexity per unit of code. For example, two programs might each keep records on the locations of various particles or units. One might do so in a simple inventory control program; the other might employ a set of differential equations to model the three-dimensional diffusion of gaseous particles. The latter would probably be the more complex program, even though the former might be longer in terms of number of program statements.

Now one could easily imagine an experiment, designed to cover reasonable points in the above variables, which employed (1) three programming languages, (2) two terminal situations, (3) two other information sources, (4) two program application areas, (5) two sizes, (6) two styles, (7) two degrees of test difficulty, and (8) two degrees of complexity. This experimental design would require that data be taken under the number of different conditions which is the product of the numbers just listed; namely, 384.

If one decided to observe, as a reasonable amount, 12 hours of programming activity (presumably assigned to different programmers in such a way as to preclude the creation of other problems in interpreting the results) in each condition, then one would require 4,608 programmer-hours of work to be observed in the experiment.

This amount of work (not to mention the equally formidable problems in setting up all those conditions, and in scheduling them) would make the experiment far too expensive to be seriously considered.

The practical problem, then, is one of boiling down . . . : refining the variables into a tractable number, and then converting that number into an experimental design. (As noted below, the formal experimentation was supplemented by further observations.)

During the course of the first four months of the project, personnel from Electronic Systems Division advised those from AMS regarding the languages of greatest interest to the government; and AMS personnel used the findings from the literature, plus the results of preliminary observations and interviews, to refine the other potential experimental variables.

As a result of this work, the following decisions were made.

(1) It was decided early that, for both the Terminal Arrangements and Conceptual Groupings studies, the languages of primary interest would be PL/1 and FORTRAN. For Terminal Arrangements, experimentation was restricted to FORTRAN.

(2) Based largely on its prominence in the literature, an aspect of style was chosen as an independent variable. That aspect was degree of modularity in the programs used.

It was noted that this variable tends to be correlated with degree of structure, and with the extent to which a software system is organized in a hierarchical manner. That is, highly modular programs are compatible with so-called "structured programming," although modular programs can be produced without following the rules of structured programming (Liskov, 1972).

(3) Based largely on the first four months' interviews and observation, attention was also focused on the rather broad variable of "choice complexity" prevailing within the program; or, more precisely, of the per-unit complexity characterizing the software being maintained.

In other words, modularity and complexity represent the independent variables, whose effects were to be studied. The dependent variables, to be observed and quantified as much as possible, related to programmer performance and efficiency.

The method for quantifying programmer performance was not fully developed until after the completion of a number of preliminary observations. No method of quantification was really satisfactory; a number were considered.

Number of modifications, per unit of time, was rejected because of differences in the types and sizes of modifications to be made.

Number of entries, per unit of time, was studied at length and rejected because of the problem of defining "entry." A long series of numbers might be just as legitimate a single entry as one character in a statement. Also, some entries might be made in almost random experimentation by a confused programmer.

A crude and subjective method, which was used, involved the programmer's opinion, which was obtained at the end of a session at the terminal. Some programmers (notably, the one with the most formal education) spontaneously made comments at the end of each session, saying things like "Well, that should take care of that"; or, "I don't know what happened there." If a programmer volunteered such a comment, he was not asked about his progress. If he did not, he was asked, "Well, did it (glancing at the terminal) let you do what you expected to?"

Comments were coded in terms of "good" for a clear yes, "so-so" for an indication of something like half-way satisfaction, and a "bad" for a clear no.

A more objective method, which was used, involved the concept of "getting stuck." The maintenance programmer was said to be "stuck," and "spinning his wheels," if he did any one of the following things:

(1) . . . sat at the terminal, taking no control actions, with nothing new coming from the computer, for at least 45 seconds.

The time threshold of 45 seconds was selected on the basis of (a) previous results showing a somewhat comparable median study time of about half a minute (Overton et al., 1973); and (b) its seeming reasonable as judged from preliminary observations.

(2) . . . went away from the terminal to study a reference manual or other general documentation, or otherwise "left the field."

(3) . . . without expressed or apparent purpose, paged rapidly through a listing or files for at least 90 seconds (which is twice the no-action threshold).

To cast this measure in a more useful and positive form, the "productive time," or time before getting stuck, was noted. (If a programmer did not get stuck in the last M minutes before ending a session, this was noted as M-E, where "E" stands for "End.") Distribution of productive time was then used to study programmer efficiency under the experimental conditions.

2.2.3.2 Design of Experiment. Given the above, important decisions, the design of the plan for the experimental observations became a relatively simple job. In order to represent more than one point on each of the independent variables, but at the same time keep the total experimental time within the constraints set up by the limited availability of experimental programmers, a simple "two-by-two" design was developed. It is summarized in Figure 1.

Ideally, only two programs would have been used in the experimental design: one of high intrinsic per/unit complexity, and another, comparable in length and every other respect, of low per/unit complexity. Then two versions of each program would have been written: one of high modularity, and another low-modularity version. Finally, the four program/versions would have been used in the corresponding four "cells" in Figure 1.

Experimental conditions	Low modularity condition	High modularity condition
High complexity condition	Results	Results
Low complexity condition	Results	Results

Figure 1. Terminal Observations Experimental Design

Some very practical reasons prevented this ideal from being met. For one thing, special programs, not connected with the real daily work of the maintenance programmers, would have meant that they were working in a very artificial, "test-tube" type of situation of which they would have been painfully aware; so the results may have been unrealistic and suspect. Other reasons were essentially economic: In effect, four new programs would have to have been developed, tested, scheduled, etc.; and the money was not available for this kind of effort. As a result, real programs, currently being maintained at the test sites, were selected on the basis of their being comparable in aspects other than complexity and modularity.

The general working environment of the maintenance programmer is this: Essentially, he is tracing the program's manipulation of data--ranging from bits to files. His purpose is to "understand" the program's modes of manipulation, but to understand them only to the limited extent necessary to let him attain his narrow goal: to make a successful modification.

The modification is deemed successful if it (1) "works" in its own right--does what it was added to do; and (2) does not "foul up" or cause new errors in other parts of the program.

In doing this maintenance, the programmer is using a graphics terminal connected to a time-sharing computer system. Within the constraints of the screen and the system, the programmer can look at input data, the program (whose length is in the low thousands of statements), and at files being created by the program.

The dependent variables (or, in terms of Figure 1, "results") center around the rate at which the maintenance programmer makes progress toward his goal. These were measured in terms of the subjective reports by the programmer, and in terms of more objective measures such as pace of work and frequency of significant problems and long interruptions.

2.2.3.3 Other Observations. AMS emphasizes that the experimental design in Figure 1 focuses on only a small portion of the interactions and variables which were listed prior to Figure 1. More crudely speaking, that experimental design illuminates only a small portion of the field of maintenance programming.

To expand that "portion of the field," informal observations were made regarding variables and factors not covered in the formal design. This observation covered all of the "cells" in the design, plus a large number of instances of maintenance which were not included in the formal design.

The advantage of the experimentation, of course, if its potentiality for yielding objective results. The advantage of the other observations, which may fail to produce significant data, is that they may cover a larger and more realistic variety of considerations.

2.2.4 Execution of Method

Observations, by AMS personnel of other companies' employees in maintenance programming, were conducted at the following locations. (They are listed in approximate decreasing order of the extent of the observations and their value to this project; this ranking does not imply any judgments regarding the intrinsic merits and abilities of the companies.)

(1) General Dynamics/San Diego

(2) Copley Computer Services, Inc.

(3) University of California at San Diego

(4) Diaspar Data Services, Inc.

(5) Professional On-Line Systems, Inc.

(6) University of California at Irvine

(7) Basic Four Corporation

(8) McDonnell Douglas Automation Co.

(9) TRW Systems

(10) California Computer Products, Inc.

Not counting travel, time for analysis of notes and tapes, etc., the "observations" included

(1) approximately 40 hours of interviews;

(2) approximately 48 hours of observations which were not videotaped, or for which the videotapes were not usable;

(3) 8 hours of videotaped experimentation.

Videotaping was done on a Sony 2000-series recorder. It was later viewed, for analysis, on a compatible player borrowed from the Capistrano Unified School District.

Notes and records were analyzed in AMS offices.

Most of the notes consisted of handwritten abbreviations which were barely intelligible to anyone except their authors. To illustrate the notes, one segment was "interpreted" and typed. It is reproduced as Figure 2.

2.2.5 Theoretical Comparisons

Independently of the execution of the experimental method, the relevant published literature was again reviewed to see what results would theoretically be expected.

2.2.5.1 Purpose. The purpose of this phase of the project was to supplement the actual, experimental results. Because of the necessarily limited scope of the formal experimentation, it was deemed especially important to compare them with whatever theoretical predictions of the results the available literature might permit one to make.

2.2.5.2 Procedure. While there almost are no "hard" data on the problems of maintenance programming with simple-versus-complex and modularized-versus-other programs, data do exist on problem-solving in a variety of other situations. Those situations which were the most analagous to maintenance programming were sought, and the findings on those situations were reviewed.

O: Today is 11 April. This is the same program as 9 April.

P: This file is too large. I'm going to reduce it.

O: He sees that it is too large from the hard copy.

P: I'll get the source file, make an absolute overlay, and hope the loader will fit it. But the loader is large.

P: I'll reduce it by two changes.

O: Both figured out at his desk. I wonder where is the best place to think.

P: I'll remove the data file.

O: And also use the absolute overlay. That's the second change.

O: He's trying to load something from the terminal.

C: Uses LDSET to locate the plotting software.

C: Goes past a time limit, cites limit, stops.

P: I'll print OUTPUT to get a listing.

O: For future study.

P: I'll catalog this file and go to the editor to enter this control card stream.

O: He enters nine lines and then an = symbol. A lot of characters for a man to remember.

C: Attaches file. (Computer is slow due to one large job.)

O: What are you looking for?

P: The thing that tells the size of the program.

C: Fills screen, then re-formats display.

P: Types '450' to list programs.

P: Damn! It's larger than the other one!

P: I'll go back to the other change.

O: Taking out some data file.

P: I'll tell it to keep my personal file . . . not the user's file.

O: When he bends over to the side to look at the listing, his head blocks out the video.

P: Check against these constants. . . .

O: Using variable names for clarity.

P: I'll dimension another variable.

O: To reduce the file size.

P: This makes the previous patch more efficient. That's the patch which tells it not to plot a curve between two points.

O: Only more than two.

Figure 2. Example of Notes Supplementing Video Tapes of Terminal Arrangements Study

Key: O = Observer's impression
P = Programmer's comment
C = Computer or machine action

3. GRAPHIC TERMINAL STUDY RESULTS

3.1 Opinions from the Field

The polling of opinions, from a technical version of George Gallup's samples, was not the intent of this project. Nevertheless, collection of some opinions from maintenance programmers and their supervisors was a convenient by-product of the "Other Observations" cited in the methodology. Some opinions, which are more or less consensual, are reported here.

3.1.1 Need for Tools

Maintenance programmers say more and better tools for terminals are needed, especially in the graphics area. Tools mentioned as desirable included support routines to insert lines, check format, check input, etc.

Available tools in these areas were criticized on at least two counts: Documentation of the tools is poor, so that a new programmer tends not to use them; and, the tools are not sufficiently standard and transportable; they are too restricted to specific devices and installations. (This latter complaint was made with particular strength.)

Better paging and file search routines were also requested. On the basis of the present authors' observations, these would indeed have been used extensively.

Difficulties were cited in getting technical data from manufacturers about specific hardward features, such as the treatment of interrupts, of concern to programmers.

Some programmers said that the program documentation for the users was generally better than the documentation for the maintainers.

McDonnell Douglas Automation Company had developed, and was beginning maintenance on, a large, interactive graphic diagramming system. Because of problems with available tools, their personnel had developed many of their own tools; included was a rather powerful text editor.

An available text editor used in three other companies was indeed powerful, but programmers said it was inconvenient to use. Actually, it is probably even more inconvenient than the programmers realized; they had adapted to some of its peculiarities. But programmers in different companies seemed to be using different subsets of its features, perhaps because each installation became acquainted only with the features which happened, at first, to seem most useful to their "shop."

At Copley Computer Systems, Inc., "edit tables" were under development. These will allow a programmer, beginning a "run," to specify which of a group of features he wants included in the run. The tool will contribute to computerized report-writing; it will allow the programmer to generate alternate modified report formats more easily, and thus to get easier confirmation from the user that the modification meets his requirements.

One installation had a graphics package, designed to help engineers design and draw electronics circuits, which included a "zoom" option: It magnified a portion of the display. The option was intended primarily for the engineers, and the programmers said they rarely used it in maintenance work.

The tool (if it can be called a tool) which seemed to be used most frequently was simply the provision for the setting of break-points in applications programs.

3.1.2 Preferred Procedures

A manager at Copley said that their on-line system made maintenance work go about ten times as fast as it had on a previous batch processing system. To the present authors, who of course studied the system there in much less detail than did the manager there, the Copley system and procedures did indeed seem extremely efficient.

Emphasis was placed on "sections" of code, each of which performed one and only one function.

The sections were quite small. Some were only two lines in length; the mode seemed to be about six, and the maximum was said to be 40 or 50 lines of code.

Because of their small size, the sections were particularly effective with an on-line system, because . . . : (1) A section is small enough to be keyed into the terminal quickly. (2) It can also be output quickly.

(3) Since it performs only one function, the maintainer can debug it separately. (4) It is easy to find the section which contains an error if its function is not performed correctly.

Documentation "is always done before and during the process of developing the program--little, if any, remains to be done afterward."

The Copley computing center uses no cards at all. At another installation, the use of magnetic tape rather than cards was advocated to facilitate maintenance. The tape was said to give more rapid feedback to the programmer. It also made it easier to keep different versions of a program distinct and well identified.

3.1.3 Where People Think

An analyst at Copley said they "very seldom write anything on paper" before keying statements into the terminal. In other words, they think at the terminal.

Exceptions were said to be "tricky little routines." These were analyzed at the programmer's desk before he turned to the terminal.

At other installations, the general picture was quite consistent. The normal mode of operation is for a programmer to do the most difficult thinking and analysis at his desk; he plans to do more routine work at the terminal.

With complex programs, he is not always able to follow his plans: Sometimes he faces challenging problems at the terminal. But the general tendency (whatever the reason may be) is for programmers to assume that they will do their deeper thinking at their desks.

Plans for that thinking may come from the terminal. At Copley, a programmer, who said he did no handwriting at the terminal, nevertheless made notes on a program listing to help him plan further changes. This was also a common practice elsewhere.

3.1.4 Other Opinions

After the experimentation (reported in the next section), or when talking with programmers not involved in the study, we raised the subject of programming languages. Their relationship to the difficulty of maintenance was of special interest.

A surprising opinion was that BASIC should be more widely used. Its many default options, particularly on number formats with which the maintainer may not be familiar, were cited as a primary reason. One programmer predicted that the Digital Equipment Corporation full-compile BASIC will be a dominant language within five years.

Finally, it was noted that beginners are sometimes a little afraid of the terminal. A similar point was made in a magazine article about reactions to an interactive graphics system. The article declared that people often " . . . felt they were in a race with the computer. It was as if, when they looked at the terminal screen, they saw all the way through to the central processor unit and a million dollars worth of electronics staring back at them, which they felt obligated to use efficiently." (Franklin & Dean, 1974, p. 9.)

At some installations, we saw none of this. For example, the experienced programmers at Copley and at Diaspar Data Services seemed quite comfortable at their terminals.

3.2 Results from Experimental Design

The data from the Terminal Arrangements experimental design, supplemented by the theoretical comparisons, produced these areas of findings:

(1) Of the experimental design variables of degree of modularity and per-unit program complexity, complexity seemed to have the greater effect on ease of work. The programmers' pace of work was most regular in the high-modular/low-complexity condition. Other factors, such as the availability of tools, may often be at least as important as modularity.

(2) For complex programs, an extremely high degree of modularity could be recommended on the basis of these results and findings from the literature.

These areas of findings are described further, below.

3.2.1 Effects of Variables

The overall results were not surprising: Complex programs are harder to maintain, and so are programs of low modularity. Later discussion will indicate that the full picture of the effects of such variables is probably broader than that given by the narrow experimental design results.

3.2.1.1 Results. A clear form of the results is presented in Figure 3a. Here the numbers are percentages; the overall average productive time for all conditions represents 100%, and the average productive time within each of the four "cells" is stated as a deviation from that overall average.

The table is self-explanatory. For example, it can be seen that most of the effect due to modularity appears in the low-complexity condition. (As will be noted later, this is contrary to what one would theoretically expect.) Similarly, the greatest difference is found between the conditions of high-complexity/low-modularity and low-complexity/high-modularity. (This was expected theoretically.)

The percentage data in Figure 3a are derived from the productive time data shown in Figure 3b. This latter figure also groups the "raw data" for productive time

Experimental condition	Low modularity	High modularity	(Combined) modularity)
High complexity	-27	-23	-25
Low complexity	+ 6	+44	+25
(Combined complexity)	-11	+10	zero*

Figure 3a. Percent Increase (+) or Decrease (-) in Productivity Under Different Conditions

(*Percentages are based on combined average under all conditions.)

Experimental condition of session	Programmer's evaluation (1st session)	Productive times observed	Programmer's evaluation (2nd session)	Productive times observed
Low complexity, Low modularity	so-so	6,14-E*	good	5,12,21-E*
Low complexity, High modularity	good	42-E	good	8,23-E
High complexity, Low modularity	bad	4,11	good	10,15-E
High complexity, High modularity	bad	6,4,9	good	3,20-E

Figure 3b. Programmer Evaluations, and Productive Times in Minutes

(*"E" following a number indicates that the session ended with the completion of that productive time.)

together with the maintenance programmer's evaluation (in terms of "good," "so-so," and "bad") of his success in the session at which the data were taken. General agreement is seen between the productive time data and the programmer's independent evaluations.

Because of the probable abnormal distribution of time data, and because of the small sample size, conventional tests of statistical significance (such as analysis of variance) were not performed on the data in Figure 3b.

3.2.1.2 Discussion. The lack of a very strong effect due to modularity, at least for the ranges of modularity and other conditions involved in this study, is interesting in the light of other opinions and findings. Interviews with maintenance programmers, reported earlier, indicated that many programmers would like to see more tools, and more standardized tools, for use at graphics terminals. Some programmers praised modularization, but a larger number--perhaps only because they tended to think of tools to help them, rather than styles to be imposed on them--called for better software tools. (What is "better" is often a matter of opinion; there are few studies, like the experimentation with Conceptual Groupings, which actually indicate that a tool improves productivity.)

Another survey independently confirmed the widespread opinion that tools have a relatively great effect on programmer productivity (Scott & Simmons, 1974). In particular, tools were generally believed to be much more important than programming which is free of GO-TO's, and which is developed in a style which should be compatible with high modularity.

In summary, the practical effect of modularity is a complex function of the exact degree of modularity in interaction with other variables of the program, language, and supporting environment.

3.2.2 Comparison with Theory

In brief, the theory (from the literature) led to three predictions:

(1) that people have a general predilection for hierarchical memory structures in themselves, and that this predilection should make it easier for programmers to work on the more modular programs;

(2) that simple programs are intrinsically easier to modify than those representing complex phenomena; and

(3) that the advantages of modular programming should be greater with complex than with simple programs.

The actual results tended to support the first two of these predictions. The third prediction was not supported; this disparity between theory and experience may have arisen because a different range of "modularity" may be needed for programs of high per-unit complexity.

The following paragraphs describe (1) the theoretical background, (2) the making of the above predictions, and (3) discussion based on further comparison with the actual results.

3.2.2.1 Background. The mathematician Henri Poincaré was not a good chess player; his memory was not good enough! He explains why his memory was nevertheless adequate for his work as a mathematician of the first rank:

> "A mathematican demonstration is not a simple juxtaposition of syllogisms, it is syllogisms placed in a certain order, and the order in which these elements are placed is much more important that the elements themselves. If I have the feeling, the intuition, so to speak, of this order, so as to perceive at a glance the reasoning as a whole, I need no longer fear lest I forget one of the elements, for each of them will take its allotted place in the array, and that without any effort of memory on my part. It seems to me then, in repeating a reasoning learned, that I could have invented it." (Poincare, 1913)

Poincare's quotation illustrates a rather universal limitation, and the typical way in which it is solved. The limitation is the limited nature of what psychologists call "immediate" or "short-term memory." This limitation is overcome by the use of some kind of plan or overview of the complex problem or program; then the problem can be worked, or the program can be manipulated.

A review of the extent of the limitation on immediate memory, authored by a user of computers, has recently been published. (Simon, 1974)

There has been much theorizing about the logical organizations which people build up in order to get around the limitation. These theories have generally emphasized the importance of models, plans, or unifying organizations to the material or program to be "exercised" mentally by the worker. The basic idea is that the person has to have some representation, map, or plan which he can use in developing his strategy regarding the problem. One theorist called this special kind of plan a "schema:"

> "'Schema' refers to an active organisation (sic) of past reactions, or of past experiences which must always be supposed to be operating in any well-adapted organic response. That is, whenever there is any order or regularity of behavior, a particular response is possible only because it is related to other similar responses which have been serially organized, yet which operate not simply as individual members coming one after another, but as a unitary mass. Determination by schemata is the most fundamental of all the ways in which we can be influenced by reactions and experiences which occurred sometime in the past. All incoming impulses of a certain kind, or mode, go together to build up an active, organised setting: visual, auditory, various types of cutaneous impulses and the like, at a relatively low level; all the experiences connected by a common interest; in sport, in literature, history, art, science, philosophy, and so on, on a higher level."
> (Bartlett, 1962)

These "schemata" may be organized in various ways, and the natural tendency seems to be (1) to prefer a hierarchical organization, and (2) if possible, to impose a hierarchical organization on the material to be remembered, to make it congruent with that which is easiest for people to work with.

A special case of hierarchical organization is G.A. Miller's (1956) "chunk." According to Miller's theory, a number can be a "chunk" of information, as can a word, or a simple visualized design. But the "chunk" can be organized of smaller elements with which a person happens to be familiar, and these smaller elements can themselves contain a great deal of information. For example, one can remember a telephone number of seven decimal digits almost as easily as one can remember a number made up of seven <u>binary</u> digits; and yet the theoretical information in the seven decimal digits is much greater than that in the seven-digit binary number.

Clearly this natural mode of thought can be the basis of a hierarchy. Consider, for example, the FORTRAN statement,

```
    IF(C .EQ. 1.Ø) GO TO 2Ø
```

For a person just learning FORTRAN, each word or symbol, including each parenthesis, might be a single unit of information. For an experienced programmer, however, a single concept could be represented by

```
    IF( ________ ) GO TO __
```

which represents one hierarchy above the beginner's level.

For a person who was somewhat familiar with the program, a single "chunk" (meaning, in effect, "Test for 'C'") would be

```
    IF(C .EQ. 1.Ø) GO TO __
```

Finally, the programmer most familiar with the program might think of the test and its consequence in "2Ø" as only one "chunk" of information.

According to theory (e.g., Mandler, 1968), a person retrieves information from his own mental memory in terms of such chunks. So, the larger the chunk which is retrieved (or recalled), the more potential information it contains.

There is an important and practical corrollary to this kind of theory. It is that people actually think *faster* when problem material is organized in terms of a hierarchy with which they are familiar.

Experimentation with words and English-language concepts (but not, unfortunately, with programming languages) indicates that this corrollary is valid. For example, it was found (Collins & Quillian, 1969) that reaction times were longer to statements mixing high-level and very-low-level concepts; and, reactions were faster to statements made up of concepts at adjacent levels on a natural hierarchy.

(For example, reaction times for statements like "A canary has wings" were longer than for statements like "A bird has wings," because the concept of wings is a part of the general "chunk" of birds; it is not unique to canaries. Consequently, "wings" is stored economically with the concept "birds," rather than with the subordinate "canary." Retrieving "wings" as a property of "canary" requires going first from "canary" to the superordinate "birds," and then to "wings." The extra step takes extra time.

(On the other hand, "yellow" is part of the "canary" concept, and people can rapidly deal with the statement, "A canary is yellow." It requires no extra steps up and down natural hierarchies.)

A much older study (Bousfield & Sedgewick, 1944) showed that people, asked to give continuous associations to a general concept, would say a burst of words, pause, and then say another burst of associations. For example, if the general category was "birds," a person might first say "hawk, eagle, vulture," (wild birds), pause for a few seconds, and then say "chicken, turkey, duck" (domestic birds). Apparently common categories are also commonly subdivided, and one thinks in terms of one subdivision until one happens to move to another one.

Another study (Tulving & Pearlstone, 1966) showed that failure to recall many things is due to failure to "get into" the category or subdivision in which the things were mentally stored. For example (referring to the previous FORTRAN example) failure to recall "C" might be caused by failure to recall the test in which "C" was used. The study also indicated that the presence of a <u>cue</u> may be necessary to moving from one category to another; a reference to test "X" might lead to the recall of "C."

Further evidence of categorized recall is provided by analysis of the common "tip-of-the-tongue" phenomenon. When one is trying to recall something that one <u>should</u> know, one tends to recall things from the same category, and even items which sound like the unknown item. (Brown & McNeill, 1966) Thus it has been proposed that the predilection for categorization causes people to supplement semantic and logical categories with phonetic and acoustic ones. (Bower, 1967)

In a previous study by the present authors, it was reported that COBOL programmers make practical use of physical similarity. In COBOL, a "paragraph" may be repeated, almost exactly, several places in a program. If a maintenance programmer finds three or four lines which he does not understand, he may look around to see if the same or similar lines are used elsewhere. There, in a different context, he may be able to understand how they function as a unit. He can then use that knowledge in the place where he was originally puzzled.

3.2.2.2 Predictions. The most germane findings from such studies can be summarized briefly:

(1) When a person has to recall old facts related to new statements (whether program statements or otherwise), he often seems to use a set of categories in the recall process; and the categories are often arranged hierarchically.

(2) People tend to use their own,"natural" category system, but they will also accept and use categories imposed by the people giving them directions. (Wortman & Greenberg, 1971; Tulving & Pearlstone, 1966) In either case, hierarchical category formation is an economical "plan" for storage and retrieval of information, and it may be a necessary condition for "keeping in mind" a large set of related statements. (Miller, 1956; Mandler, 1968)

(3) The superordinate/subordinate relationship is not the only possible basis for categorization. Many other organizational schemes may be fitted to a hierarchical plan. These range from acoustic similarity (Bower, 1967) to the mental treatment of a few lines of code as falling in the same functional class as similar lines elsewhere.

In other words, thinking man apparently has a predilection for hierarchical structures for storing and retrieving concepts, which are in turn defined by various categories. This may be the only way people can handle large amounts of information or large numbers of statements at a time.

Given this theoretical background, the following predictions are a logical conclusion:

(1) Highly modular programs are more congruent with the human predilection for hierarchical and categorical memory structures than are programs of low modularity; therefore the modular programs should be easier to maintain.

(2) Programs of low per-unit complexity should also be easier to deal with categorically, and they should be intrinsically easier to maintain.

(3) The advantage of modular programs should increase with the complexity of the program.

In other words, modularity should help the maintenance programmer to keep relevant aspects of some one else's program in mind. Perhaps more important, it should also provide him with a sort of "index" to the program, to help him focus down on the points at which code must ultimately be laid.

The universality of hierarchical and other categorizations of concepts, illustrated in a variety of experiments of the type cited here, indicates that people have a strong bias for using this kind of "plan." (The bias may be innate, or it may be due to the kind of education and experience that people normally receive. But its origin is not germane here, as long as the bias exists.) People, including programmers, function best when the material they have to work with fits their biases, habits, and cognitive approaches to the world. A computer program organized in a quasi-hierarchical fashion, or one with distinct and realistic categories such as modules, should be easier to comprehend, according to the theory, than one not so organized.

3.2.3 Modularity Recommendation

Perhaps the most important divergence between the theoretical predictions and the actual results concerns the effect of "high modularity" in interaction with "complexity." According to the theory, modularity should be of greatest help in connection with the greatest degree of per-unit complexity. In actuality, this difference did not appear.

In the opinion of the present authors, this divergence probably stems from the fact that the experimental design permitted only two degrees of modularity: "high" and "low." In particular, "high" was probably not high enough to allow the effects to appear.

This opinion is based in part upon the results of a previous study (Overton et al., 1973) in which modularity was also an independent variable; but in which it was found that the experimental programmers preferred a higher degree of modularity than any which was actually represented in the study. (<u>Smaller</u> modules and submodules represent a <u>higher</u> degree of modularity.)

It is also based on the academic literature regarding problem-solving. While the artificial problem-solving studies may not be entirely applicable to maintenance programming, they do show that, for most of the categories of statements with which people actually work, the category size is rather small.

Most importantly, however, the authors cite the "Studies in Conceptual Groupings" described later in this report. (Previous, similar studies are also relevant.) In the studies covered here, maintenance programmers were helped quite significantly by automated displays of groupings of types which programmers tended to deal with as units. And the modal sizes of the helpful groupings fell in a range of approximately six to eleven statements. These are, of course, much smaller than what would normally be defined as the smallest module in a large software system.

An incidental implication of the present studies relates to the question of the maximum number of programmers who should be assigned to the development of a module. (While the present studies are focused on maintenance, many of the lessons probably apply to development.) After citing other recommendations of a maximum of about ten programmers to a module, one author (Simmons, 1972) declares that his review indicates that the number should be a maximum of six people.

The opinion of the present authors is more extreme. In view of the lessons learned in these volumes, modules should probably be broken down, in the planning phase, to sub-modules so small that the maximum number of programmers assigned to each is <u>one</u>.

3.3 Other Experimental Results

A number of other conclusions can be drawn from the notes and data from the Terminal Arrangements activities. These include the following:

(1) The summed effect of small distractions may be greater than most people have suspected.

(2) The programmer is more likely to make mistakes when he "shifts gears" and goes from one level of discourse to another.

(3) Appropriate visual analogs of digital displays might facilitate people's maintenance work.

(4) Programmers often ignore their terminals, during or at the end of a session, for considerable lengths of time. These lapses may indicate that the terminal environment is not conducive to some necessary kinds of thinking.

(5) Allocation of tasks between the programmer and the operating system could be improved by having the system perform even more identification and clerical support functions.

(6) For information to support his work, the maintenance programmer at a graphics console calls on an impressive variety of sources of information.

(7) Sometimes a program module must be analyzed as a whole but, because of its size or other problems, it cannot be displayed as a whole at the terminal.

(8) Several other points can be made. These include the possibilities that (a) characteristic turnaround time, or system response time, influences the type of errors which the programmer characteristically makes; and (b) different human senses interact in ways which may be significant.

Potentially, all of these findings have practical implications; but some will be easier to implement than others.

The above items are discussed in greater detail below.

3.3.1 Distractions

System designers seem to have a tendency to want to display everything which a programmer might ever want to see on a graphics terminal. This tendency is understandable in absence of definite specifications regarding what he will need to see. But it may constitute a source of distractions, and there are indications that distractions in general are a very definite handicap to the maintenance programmer.

Of course distractions can come from many sources. External noises and sights are the most obvious. But they can come from the very activities which the graphics system requires the maintenance programmer to perform. In one case, the programmer was checking the functioning of a change, and he was required to resubmit accounting-type information before making the "check-out run." But apparently this information reminded him of costs, which in turn reminded him of a core utilization problem, which caused him to forget the exact points he needed to check in the run he was starting to make.

Displays which were not under the programmers' control, such as lengthy displays inserted by analysts for the possible monitoring of the operating system, seemed particularly distracting.

According to Dr. John Goodenough (in a personal communication in connection with an earlier project), maintenance programming is more difficult and more "subtle" than development programming; the maintainer must worry about more indirect problem areas, with which he is less familiar, than the developer. Accordingly, he is more vulnerable to disruption when he is in the midst of an analysis of a program section which may seem "tricky" to him.

Analyses have starting points; and, in programming, these are often displays. In the words of an older discussion of problem-solving, "Trains of thought are pulled by stimuli." (Overton, 1959) Paraphrasing those words, delicate trains of thought can also be derailed by irrelevant stimuli.

When the effect is measured carefully, even very small visual displays can have a significant effect on work. Inspired by the theory that one adapts in almost a physical sense to what one is observing, a test was made of the effect of a single, irrelevant flash of light. (Helson & Steger, 1962) The test showed that it retarded human performance. Later experimentation (Vruels & Schmidt, 1966) verified the effect and suggested a reason: "The visual system seems to act like a circuit with two or more integrators in series, but with a bypass loop that allows a partially unprocessed signal to modify slightly the already processed signal in a comparator." In other words, the irrelevant signal may cause the first one to be distorted while it is, figuratively speaking, en route to mental processing.

Because of the complexity of the process of maintenance programming, it was not possible to obtain accurate measurements of the times that programmers were distracted, or the seriousness of these distractions. The programmers themselves probably would not know. One can be distracted without being aware of the effects or extent of the distraction. (Also, it would distract you to ask you if you were distracted!) In spite of the lack of objective data, however, the authors developed the strong opinion--which is reinforced by the limited but relevant experimentation of the type cited above--that the summed effect of small distractions in maintenance programming would be a surprisingly great waste of programmers' time.

3.3.2 Level Shifts

Many small phenomena, like blinking one's eyes while reading, are frequent but not conspicuous. When these phenomena affect one's work, however, their cumulative effect may be considerable. An example are the phenomena associated with shifting between "levels of discourse" (i.e., levels of description of a program, mode or language of statement of the program, etc.) in maintenance work.

It should be emphasized that some such shifts are necessary. The dissertation of Brooks (1974) cites evidence behind the theory that programmers necessarily and frequently shift up and down between "planning" (on a small scale) and coding. (In a personal communication, Brooks also cited a spontaneous experiment with Prof. Djikstra, who is generally thought to be a firm advocate of top-down programming. Brooks said that when Djikstra's programming was analyzed by Brooks' system, Djikstra also was found to move up and down frequently between levels.)

The effects of shifts, even at the same level, are detectable. It has been found that even when a person knows two languages well, his work is slightly delayed every time he shifts between them. (Kolers, 1973) This effect was found with natural languages, but it probably applies to programming languages as well.

In this experimentation, mistakes by the maintenance programmer often seemed to accompany shifts between levels of description. Errors were made in the detailed formats of input data when the programmer had just been worrying about the higher-level logic of the program. Or, engrossed in the logic of a routine, he fails to push a button and ask for a copy, to use in later higher-level analysis, of the graph the routine is producing. Or, thinking of the files which a program does and does not use, the programmer recalls that data need to be changed in one of them; he tediously changes the data, then returns to thinking of the program functions and realizes that the program does not use that particular file!

Asked about this kind of error, one programmer explained: "Well, it's like shifting gears. You're more likely to goof when you have to shift gears."

The implication of all of this is obvious although perhaps difficult to implement: The procedures for maintenance programming at terminals should discourage unnecessary shifting up and down between levels of discourse.

3.3.3 Visual Analogs

The term "visual analog" is used here to denote a graphic (analog-type) representation of digital data. For example, the columns projecting from the quadrant in Figure 4 are analogs of the numbers, giving productive times under four conditions, in Figure 3a.

Often visual analogs are not generated for one simple reason: They cost more than digital displays. Even the simple drawing in Figure 4, for example, requires much more work than would be needed to simply write down the numbers. But the advantages may be significant.

In these studies, the programmers at graphics terminals consistently used visual analogs for at least one kind of test of the acceptability of modifications. When the programs were designed to draw graphs for engineers, a modification would rarely be tested in terms of the data files it created or the equations it used; instead, the programmer would have the modified program draw a new graph, and he would look at it and ask either himself or an engineer or both: "Does that look reasonable?"

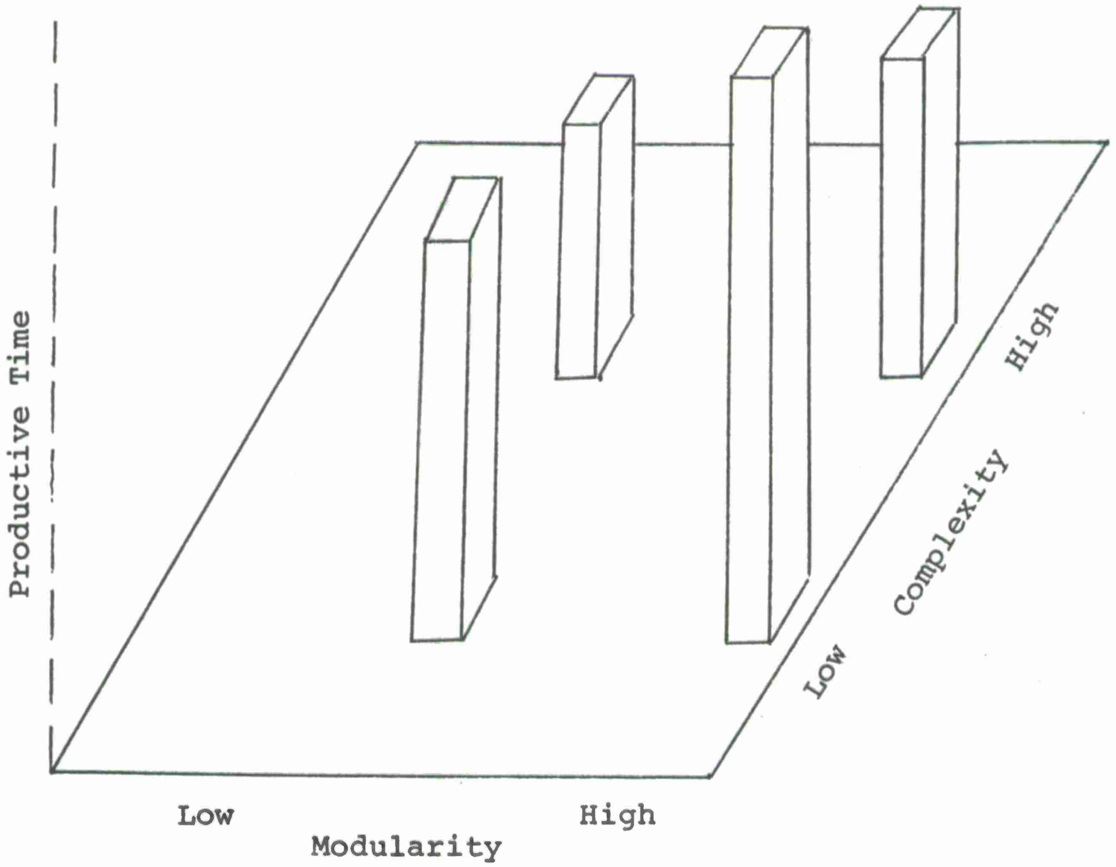


Figure 4. Visual Analog of Productive Times Data

In general it seems that when visual analogs can be produced for "reasonableness checks," the analogs indeed provide an efficient way for people to make these checks.

Visual analogs may be appropriate for purposes other than the making of gross tests of program functioning. For example, it has been shown that people are less likely to make gross errors, and are more likely to remember what they see, if they are shown pictorial displays rather than numerical arrays of a certain degree of complexity. (Newman, 1966)

Appropriateness of displays will also be determined by the people for whom they were intended. For example, it was found that two dozen English housewives performed a decision-making task better when the data needed to make the decisions were expressed in terms of rods of differing length, rather than in terms of numbers. (Hammerton, 1973) Whether college mathematics majors would have done better under these same conditions is not known. However, it was found that in a simpler job--simply reacting to the differences between pairs of numbers--college students reacted more quickly when the quantities were presented in analog form than when they were presented digitally. (Moyer & Landauer, 1967)

In brief, it is safe to say that there is a natural tendency, which may be overcome to some extent by education, for people to work more easily with visual analogs.

The maintenance programmers used another technique which was related to visual analogs. When they were puzzled, or otherwise under stress, they tended to simplify the picture before them. When a graphics package did not seem to be working properly, the programmer projected the normally three-dimensional view (generated by the program) into a two-dimensional view (also generated by the program). Apparently he felt that he could "get started better" in understanding the problem with the simpler view. Similarly, data-processing routines for dealing with many variables were "compressed" by maintenance programmers so that the routines operated on, and displayed results from, only a few variables at a time. To deal with a complex problem, the programmers clearly preferred to start with a simple picture (both figuratively and literally) and work up to a complex picture, rather than vice versa.

It seems, then, that the natural preference for visual analogs should be exploited, rather than fought, wherever it is feasible to do so in maintenance programming. Two feasible areas are probably the making of reasonableness checks, and the simplification of complex pictures as starting points for debugging.

3.3.4 Avoiding the Terminal

At one time during one experimental session, one highly-educated programmer said, "Let's go back to the listing to get our bearings." He then ignored the terminal while he studied the use of large data sets. Other programmers similarly avoided the terminal while trying to work out some kinds of problems.

This kind of behavior has been statistically verified elsewhere. Empirical data were collected on the times of terminal use in an interactive system (Boies, 1973), and a principal finding was that it was common for programmers to be " . . . leaving their terminals for relatively long periods of time . . . often (returning) only to sign off . . . ." The interpretation the present authors make of this finding is that maintenance programmers often try to work something out without using the terminal; and if they do not succeed in a reasonable time, they sign off and go to a desk to try some more.

Another part of the empirical study (Boies, 1973) reported that when the system detected an error in FORTRAN, " . . . a diagnostic message and the offending line were displayed on the terminal and the user was given a prompt to correct the error. In the 26 programs in which this occurred, only once did the user correct the errors in the program." (Underlining was added by the present authors; we would also have put an explanation mark at the end of the sentence.)

Why did the programmers not correct the errors when the system pointed them out? In some cases, apparently, they tried to: " . . . six times the system crashed or the user session ended while . . . waiting . . . to correct the program." In some cases, in other words, it was not easy to make the corrections at the terminal at that moment.

The most frequent case, however, was for the person to request the system to continue without the error being corrected. Here the programmer evidently wanted to pursue one train of thought at the terminal, and to worry about the error at a different time.

This behavior seems to contradict somewhat the claim, made by some proponents of time-sharing systems, that the graphics terminal is an aid to the programmer's thinking. But the picture is probably not that simple. What seems to be happening is this: The terminal is an aid to some kinds of thinking; so the experienced programmer tends to do those kinds at the terminal; but when he needs to do another kind, he avoids the terminal, and he looks at different things.

These tendencies have practical implications which will be discussed in Section 4 under General Interpretations and elsewhere in this report.

3.3.5 Task Allocation

In batch processing, it is customary to have the operating system write some identification on a job when it is run. Of course the user has to give his name, etc., but the software can add the date of the run and useful but non-unique data such as compile time and execution time. This custom is an example of the result of a process which, when done formally, is called "task allocation": allocation of tasks to be performed by the programmer versus those to be performed by the computer.

Task allocation is often not done formally, and the informal results vary from system to system and from installation to installation. Because time-sharing systems are newer than batch, there is probably even more variation from system to system with graphics consoles than there is with batch operations. Because of this variation, observations made at one installation may not be valid in other "shops." In any case, some observations on task analysis were made during the Terminal Arrangements studies.

Of course one always wants to assign the "dog work" to the computer. But this has not always been done. Display identification is one area in which improvements might be made.

In one case, a programmer was working at a terminal with listings of two different compilations of a program. He slipped back and forth among the fan-fold sheets, mixing them up as he tried to study them in relation to the displays on the screens. But the listings were identified only on the top sheet of each; and, in the middle, the programmer became confused. For maintenance work, a simple identifier on every fan-fold sheet might be worth while.

In a more general case, the system generated a graphics display, and the programmer wanted a copy of it for future references. He could get the copy, but then he had to identify it by hand. An easy, semi-automatic notation of identifying data--especially, parameters being changed from one "picture" to another--would ease the programmer's job.

After identifying himself, the programmer should stay identified as long as he is at a terminal. On one system, his identity is keyed to the application program with which he is working, and he has to re-submit his identity if he calls in another program such as a library routine. There are other examples of systems which do not support the programmer well in handling clerical and bookkeeping data.

Finally, options (regarding display parameters, computation mode choices, etc.) should be treated differently when a programmer is doing a series of tests on a program. Instead of having to make all of the choices (like a user using the program for the first time), he should be able to say, in effect, "Now change only option X; make it B." This would permit easier and faster comparisons; and, probably, better deductions from the series.

3.3.6 Sources of Information

When terminal-oriented systems were originally being developed, news releases spoke of glowing promises. Seated at his terminal, each of many people could have easy access to tremendous computing power and to valuable banks of public information.

This remains a worthy goal. But for the maintenance programmer, at least, the picture of him wed to his terminal, with the two of them overcoming all bugs and modifications, is a picture which is strikingly incomplete.

The fact is, that to support his work, the present-day maintenance programmer calls on an impressive variety of sources of information.

(1) The news-release picture is accurate to the extent that the one most frequently tapped source of information is probably the terminal.

(2) In almost every case, however, the programmer comes to the terminal with a marked-up print-out, usually a program listing, as a guide to at least his first actions. If he seems to "get stuck" in maintenance, and if the paging and other display routines available on the terminal do not satisfy him, he may spend most of his time studying the print-out.

In some installations, the programmer had trouble finding a place to lay out all of his listings.

(3) Depending on exactly who is involved and available, he may telephone or otherwise talk to another programmer. Often this is not the original developer of the program, but rather another maintenance programmer who has worked on it recently.

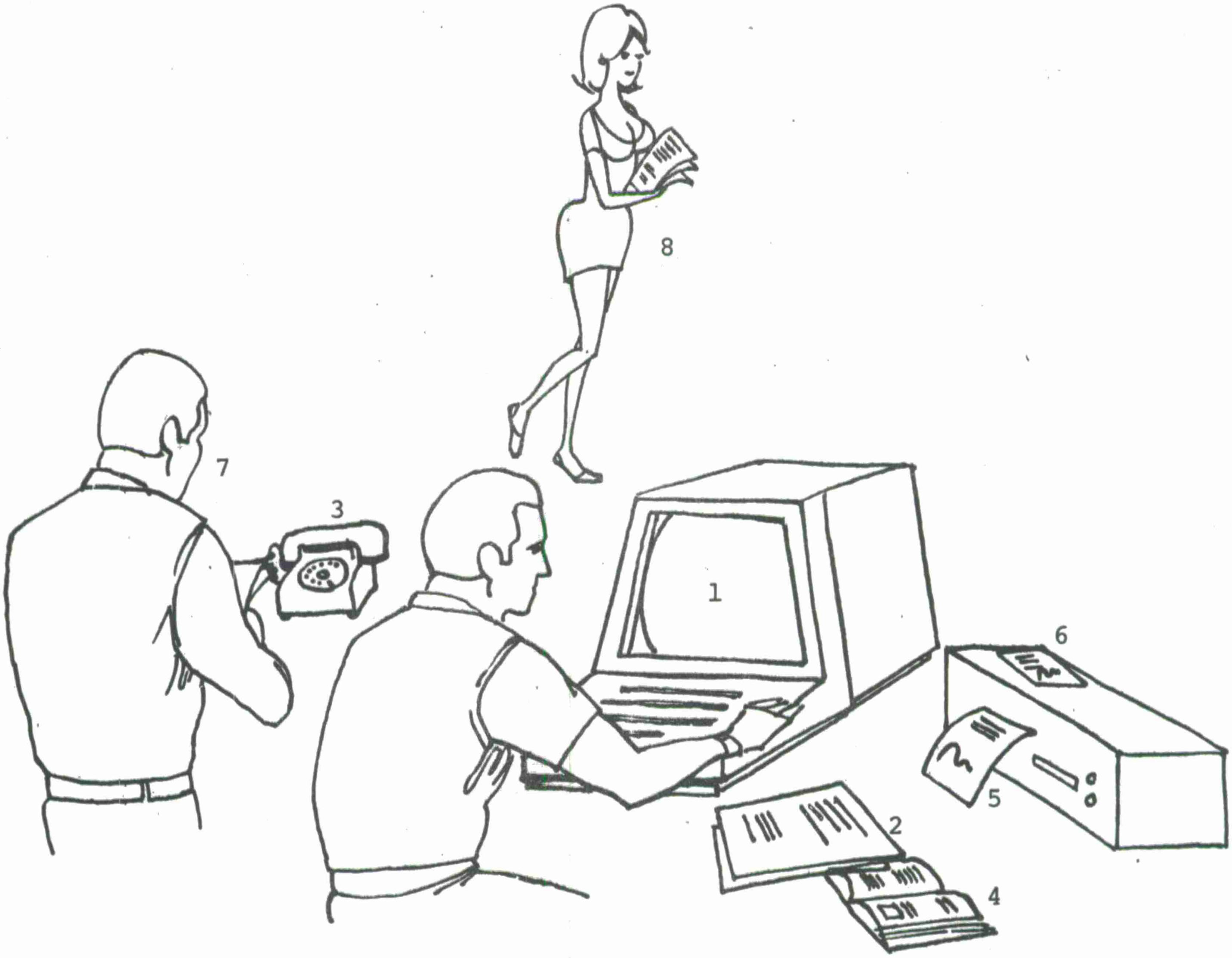
It was interesting to note that the type of question asked of the other programmer was almost always at either a high level of discourse (e.g., concerning the purpose of a graph for an engineer, or other conditions regarding the use of the program) or a very low level of discourse (e.g., concerning the purpose of a single symbol). Questions of intermediate level, such as the exact function served by a short section of code, were rarely asked.

(4) The maintenance programmer sometimes turned to the program manuals for information. But this was relatively rare.

(5) At vector (or picture-drawing) graphics terminals, the terminal is supplemented by a photocopy machine, which takes a "hard" picture of what is on the "tube." The programmer interacts with this machine by telling it when to make a hard copy.

(6) Old copies from the machine form a much more important source of information for the maintenance programmer. He uses them to compare "before" and "after" computational results, to see if he has made progress.

At no installation was there a formal assignment of a place for the programmer to store these copies temporarily. He usually just laid them on top of the photocopy machine. In one case,



.Figure 5. Terminal Arrangements

Sources of Information

he scotch-taped two of them to the edges of the graphics terminal.

(7) The engineer for whom the program was being modified, or, more generally, the programmer's customer was occasionally available to the maintenance programmer. From him, the programmer got information about the acceptability and reasonableness of tentative computational results.

(8) Distractions abounded, especially in the larger installations.

No one in the field should be surprised that the maintenance programmer at a terminal uses these and other sources of information. In planning the physical arrangements around a new terminal, however, a little more thought should be given to the fact that he will use them.

3.3.7 Module Displays

When programmers turned away from the terminal and to a print-out, they normally spread out more than one page of paper; they seemed to be studying a large section of coding.

This behavior helps indicate the existence of a rather obvious problem: As a rule, a module needs, simply because of "the nature of the beast," to be analyzed as a whole. But sometimes the module is too long to be displayed all at once, in readable form, on the terminal. Also, deficiencies in the system's paging and indexing routines can prevent some modules from being displayed as a whole.

In these cases, the maintenance programmer can turn to the corresponding hard copy, if it is available, and study it. Otherwise, he must try to use his memory or make spur-of-the-moment notes to supplement the display. In any case, he is using his own papers or memories to supplement the system.

3.3.8 Other Points

Miscellaneous points from the experimental sessions include the following:

(1) When system response through the terminal was rapid, two of the programmers (who knew they

were being observed) tended to repeat the same kind of error in quick succession.

The reason for this perseveration of errors may be that the programmers felt pressures. (As previously reported, new people at terminals often feel as though they are "in a race with the computer." And being under observation may bother some people.) It is an established principle that too much pressure tends to disrupt complex problem-solving, even though it may encourage simple, repetitive work. (Overton, 1959) (In the extreme, it would do no good for a dictator to order his subjects to "Be intelligent!" or "Be creative!") Because of this principle, some of the avoidance of the terminal cited above may be necessary. Or, programmers should not overestimate the cost pressures in terminal usage.

(2) Addition of sound to sight can result in some tricky mixtures. For example, it has been shown that if people see a series of numbers, they can remember better the order in which the numbers were presented. But if they hear the same series (over the same span of time), they can recall the exact numbers better, but they are more likely to get the order of presentation wrong. (Briggs, 1973)

If the object is to make a person remember something, this is best accomplished by requiring him to recode it to another sense--for example, by presenting visual symbols to him and having him repeat them orally. Often in maintenance, however, the burden of remembering something like a specific real number should not be given to the programmer in the first place; scratch-pad memory or a good editor would be a better tool.

As another example of an odd effect, it has been found that people *see* less on the side of their visual field which is opposite from an irrelevant noise. (Nice, 1973)

When the vibration of noise includes frequencies in the range of about 10 to 30 Hz, it is in the range which most degrades visual acuity. (Semple et al., 1971)

In brief, there are conditions in which the freedom from extraneous noise could be more than an esthetic goal, but in which it could somewhat improve the programmers' efficiency.

(3) In general, more emphasis needs to be placed on the question of what displays maintenance programmers need to see together, for purposes of comparison.

Some examples of this need are trivial: A programmer wanted to compare a print-out with the terminal, but he did not have a big enough flat surface around the terminal on which to place the print-out. Or, he needed to compare drawings made with different scalings and other parameters, and he had to scotch-tape old copies to the sides of the graphics terminal.

Others are more basic: In a previous project which included a study of "split screens" on terminals, programmers working with highly modularized programs were found to be most likely to elect to use the split-screen feature. Apparently the reason was that they wanted to compare different versions of the same module, or a module and its resulting file.

In any case, much of maintenance involves comparisons, and systems should be designed to facilitate those comparisons which are most likely to be made.

(4) The period of time in which a person can concentrate on intellectual work may be limited. No programmer scheduled himself for more than an hour of actual maintenance work on a terminal. On the other hand, more routine work was done for longer periods without breaks.

A brief survey of a related problem is provided by Broadbent (1971). Discussing research on people watching aircraft radars, he notes that errors rose significantly " . . . after . . . only half an hour; which was a remarkably rapid onset of 'fatigue.' A deterioration of this sort has been found in large numbers of similar tasks since the original investigation . . . . It is of course of some practical importance . . . ." (p. 19)

As in other cases, freedom may be the best policy: Allowing the programmer to pace himself according to the nature of the problem before him may minimize errors.

4. INTERPRETATIONS AND RECOMMENDATIONS

4.1 General Interpretations

Many programmers do not like maintenance work; it is just one problem after another. At best, you (the maintenance programmer) face the problem of poring through a problem to find the right places to make a modification . . . complicated by the problem of insuring that your modification does not "foul up" some other obscure part of the system. At worst, you have to correct someone else's failure . . . to track down a bug whose creator was not aware of creating it.

Problems can be cast in ways which make them easier to overcome. However, much prior study of the problems is necessary before better "casting" can actually be accomplished. From one point of view, the Terminal Arrangements studies are concerned with casting the problems of maintenance in a form which is less offensive to, and difficult for, programmers.

(The studies in Conceptual Groupings can be viewed in the same light, although they are concentrated more on determining the feasibility and value of a single set of techniques.)

Casting programs in modular form, the Terminal Arrangements study found, seems to make maintenance problems more tractable. The more complicated the program area, the greater the degree of modularization which may be necessary to help the programmer.

In general (going beyond modularization) there is probably a powerful and important interaction between (1) the intrinsic nature and complexity of a problem, and (2) the modes in which it is best stated and analyzed.

For example, no college professor of mathematics would attempt to present a complex equation by reading it in a one-dimensional stream of words and letters. Instead, he would write it on a blackboard, where students could look at it, read it back and forth, and study symbols from top to bottom if necessary. In other words, a writing surface is better for presenting equations than is a stream of spoken words.

In terms of the design of arrangements for terminals, the point of the little story of the mathematics professor is this: The blackboard is a form of *memory*, and it is quite analagous to a computer memory. The professor uses the blackboard to supplement *human* memory and thus to cast the problem in a more understandable form.

At terminals, as well, there are different kinds of "memories," ranging from pieces of paper, through the terminal itself, on to computer memories. The general goal is to arrange these to complement, rather than conflict with, the human memory of the programmer.

This general goal can be broken down into less general areas:

(1) For program and system design, a useful *style* is one which helps a maintenance programmer (a) to trace the structure into which a modification has to fit, and (b) to recognize the functional blocks of which the structure is built.

(2) The human memory is volatile and fragile, and the *physical arrangements* at the terminal should not require the programmer to acquire information from one source and then suffer a delay (plus possible distractions) in getting complementary information from a different source.

(3) Between programmer and computer, *task allocation* should obviously assign as much "dog work" to the computer as possible.

Of course people are adaptable. They work constantly with arrangements which are not optimum. For example, they take notes on incomplete displays of modules, or they try to remember things which, ideally, the computer would be remembering for them. In any case, they work at lowered efficiency. And if the arrangements are too discouraging, programmers will, mentally if not physically, desert the terminal.

In summary, maintenance programming poses problems. The main guideline for terminal arrangements should always be to try to configure the problem in terms of the displays and memory assignments which are most conducive to the solution of the problem. The next section will present some specific findings and recommendations resulting from the study.

4.2 Recommendations for Immediate Implementation

A major purpose of the studies in terminal arrangements was the development of recommendations of value to maintenance programming. Several such recommendations are made below.

Some of the recommendations could easily be implemented, and others would require additional research and/or development work. Also, the evidence permits some to be made more firmly than others. The recommendations are listed in the approximate decreasing order of ease or immediacy of implementation, and <u>not</u> in order of intrinsic importance.

The section references in the recommendations give the locations of fuller discussions of the topics of recommendations.

(1) If a large program has a hierarchical structure, and if a maintenance programmer is allowed to familiarize himself with the hierarchy before doing detailed work at different levels, he may "think faster" when he does begin work. (See Secs. 3.2.2.1 and 3.3.2.) Except in emergencies, such familiarization with the hierarchy should be recommended to the new maintenance programmer.

(2) Programmers should not be required to schedule terminal work for blocks of time which seem excessively long. Although the time a person can work productively at a terminal is unquestionably a complex function of the nature of the program and of other factors, there <u>is</u> a limit on the length of time a person can really concentrate on a problem; inconclusive evidence suggests that the limit may sometimes be as short as 30 minutes. In the absence of firm evidence, self-pacing of the programmer's work is suggested. (See Sec. 3.3.8.)

(3) Familiarization procedures should encourage new programmers <u>not</u> to feel as if they are "in a race with the terminal." With the "race" attitude, a programmer may repeatedly make the same errors. (See Secs. 3.1.4 and 3.3.8.)

(4) When a new terminal system or installation is planned, predictions should be made regarding the usage of hard copy and other such material around the terminal; and the physical facilities around the terminal should be sufficient for this material. (See Secs. 3.3.6 and 3.3.8.)

(5) Program version identifiers should be required on each output page (or terminal screen) to aid program search operations by the programmer. Other pertinent input data and parameters should be readily available on the operator's graphic screen or on the output page, to facilitate the maintenance process. (See Sec. 3.3.5.)

(6) People tend to make mistakes when they "shift gears," between different levels of detail, when they are working on a program. Both tools and procedures should be designed for minimizing unnecessary shifts up and down levels. (See Sec. 3.3.2.)

(6a) Procedures partly depend on available tools. But there would be benefits from simply giving the programmers examples of shift-related errors, and encouraging them to schedule their levels of work as homogeneously as possible.

(6b) Split-screen operations (allowing simultaneous display of two different routines or files) and note-taking routines (facilitating temporary storage of data needed later for a different level of analysis) are examples of useful tools.

(7) Visual analogs (defined in Sec. 3.3.3) are efficient tools for verifying the reasonableness of program functioning, both during debugging and after the incorporation of program revision. Depending on the graphics terminal and computer system being used, many such visual analogs could be developed at low cost; they should be.

(8) Modularity should be a criterion for the acceptability of large software systems.

(8a) For systems which are large, but in which the per-unit complexity is not great, each module should generally be no longer than 40 lines of code; and the module should be developed by only one (1) programmer. Such a module can be displayed in its entirety on a screen, and it can be input and output quickly, and maintained by one person. (See Sec. 3.1.2.)

(8b) For programs which are complex in terms of the implications of each statement, a higher degree of modularity (i.e., smaller modules and less dependence between modules) is recommended. (See Sec. 3.2.2.)

Note: This recommendation also applies to the area of conceptual groupings.

(9) Priority should be given to the systematic development of computer aided tools for software maintenance. (a) Those tools which are available tend to be parochial; they should be standardized and made more transportable between computer systems. (b) Further tool development should be guided by the real needs for the tools. Programmers may not really know what tools they need, but they expressed a general desire for a powerful text editor with features of (i) insertion, (ii) deletion, (iii) substitution, (iv) formatting, (v) input, and (vi) paging. (See Sec. 3.1.1.)

(10) These recommendations are based largely on observation and experimentation involving the use of FORTRAN; and, to a lesser extent, PL/1. To supplement and expand such recommendations, there should be additional experimentation and observations in PL/1, in COBOL, and in other languages.

5. STUDIES IN AUTOMATED CONCEPTUAL GROUPINGS

5.1 Introduction and Summary

Often, to find out how people approach their work, there is no substitute for listening to them talk about it. In previous studies, we listened to programmers explaining their programs to other programmers who were going to have to take over maintenance of the programs, and to tape recordings of programmers talking to themselves as they worked on others' programs. The typical talk had this "flavor" to it:

> "Now this gets you into a big old DO-loop, and it does so-and-so; and along the way you go to this other little routine which stores the results; and later you go to a print routine. . . ."

From many such spontaneous words of explanation, it is clear that program statements are arranged into groupings; these are the "natural" units of thought for analyzing the program (at least at the most common, working level of discourse).

The previous studies also noted that these small, elementary groups of program statements are not the ones the programmer typically chooses to describe in formal documentation. One difference is in size: In face-to-face explanations, groups show a median size of about 5 lines or 6 statements (Overton et al., 1973, p. 40), whereas a median three times as large (17 statements, to be exact) was found in written documentation (in an unpublished review).

It is speculated that the origin of the difference is in the intent of the programmer: In documentation, he may choose units which minimize the work of documentation; in face-to-face explanation, however, he may choose more basic and clearer explanatory units.

If these basic "conceptual groupings" could be displayed in some way, they should (it was reasoned) help the maintenance programmer significantly—help him follow the structure of the program more easily and rapidly, and help him identify the exact little sections to modify.

The demonstration of the existence of conceptual groupings, and the reasoning about their potential value, formed the background to the present Studies in Automated Conceptual Groupings.

In brief, pilot programs were developed to automate display of groupings. PL/1 and FORTRAN were the target languages. Program development was guided by the results of previous studies, supplemented by analysis of "marked-up" listings (on which programmers had scrawled brackets around small sections of statements as they read and mentally grouped them).

Experiments were conducted to evaluate the helpfulness of such programs. (Because of scheduling and other considerations, the majority of the experimentation happened to involve FORTRAN.) A principal finding was reported to be:

> "Maintenance programming efficiency, at least on moderately-commented FORTRAN programs, is increased significantly by the use of a Conceptual Groupings processor."

In view of the great expense in maintenance of FORTRAN programs (many of which are not well commented), this finding was rather dramatic.

Experimental results with PL/1 tended to be rather anticlimactic. In summary, the results with PL/1 were consistently favorable to the Grouping Program (with programmers' expressions of opinion including ". . . very, very helpful" and "Great!") but the experimental data collected were too small (in terms of numbers of programmers) for the results really to be conclusive.

In addition, there were indications that the PL/1 Grouping Program needs to be revised to (1) reflect more adequately the habits that programmers bring to PL/1 from other languages, (2) conform more closely to the structure and logic of the language, and (3) complement other tools for checking out and maintaining PL/1 programs.

In spite of these indications, the PL/1 Grouping Program seems capable of improving maintenance programmer productivity by ten or fifteen per cent. Hence, both it and the FORTRAN Grouping Program merit more development and wider usage.

5.2 Research Methodology

The manner in which a programmer conceptually groups lines of code, portions of files, etc., is influenced by the programming language in which he is working. Furthermore, the features which statistically tend to define groups, according to the previous studies, vary from language to language. More precise definition of these features, and therefore of the advantages of helping a programmer recognize the groups, must be done on a language-by-language basis.

In consultation with ESD (MCIT), FORTRAN and PL/1 were the target programming languages selected for groupings analysis. Attention was given to FORTRAN first in the conceptual groupings study. The knowledge and data accumulated from the earlier RICASM Studies were updated and extended to select the conceptual groups most likely to be consistent, easily identifiable and usable. Having made the selection, specifications were established for a groupings programs using FORTRAN and PL/1 as the target languages.

The first versions of conceptual groupings programs were to be developed because of the assumption that such programs would help a maintenance programmer recognize useful groupings, and that this recognition would facilitate his work. Experimentation with the first groupings programs would test this assumption, and permit one to say whether or not, in actual practice, such programs would be effective aids to maintenance programmers.

In general, an experimental conceptual groupings program accepts as input a higher-level-language program which can be successfully compiled. The groupings program operates as a post-processor; it creates a new listing which displays those conceptual groups that the groupings program has identified within the source program.

For FORTRAN, the experimental conceptual groupings program (designated GP-F) was designed to accomplish the following functions: (1) identify groups by "like-statement types," (2) print formats under each referencing I/O Statement; (3) sort declaratives to beginning of the programs; (4) indent nested DO loops; (5) mark transfer statements; and (6) mark I/O Statements.

The Conceptual Groupings Program for FORTRAN was coded in the symbol manipulation language SITBOL (a version of SNOBOL) because of the difficulties in using partial word operations in FORTRAN itself. A complete program description of the Conceptual Groupings Program for FORTRAN (GP-F) is found in Appendix B. The program documentation includes: (1) The general system description; (2) Functional specifications; (3) Program Implementation; (4) Flow Charts for Grouping Program; (5) Grouping Program Listing (SITBOL); (6) Ungrouped Source FORTRAN Program Listing; and (7) Grouped Source FORTRAN Program Listing.

As part of identifying conceptual groups the groupings program for PL/1 (GP-P) was designed to accomplish the following functions: (1) identify large conceptual groups; (2) identify groups by "like-statement types"; (3) assign logic levels; (4) repeat "notes"; (5) reformat declarations; (6) indent nested control groups; and (7) mark I/O, Entry, and ON-condition statements.

The Conceptual Groupings Program for PL/1 (GP-P) was coded in PL/1 so that it could later be tested in an operating environment with a manufacturer supplied PL/1 compiler. A complete program description of the Conceptual Groupings Program for PL/1 (GP-P) is found in Appendix C. The program documentation includes: (1) General System Description (GP-P); (2) Conceptual Groupings Program for PL/1—Operating Instructions; (3) Groupings Program for PL/1 System Block Diagram; (4) Groupings Program for PL/1 (GP-P) GP-P Flow Charts; (5) Source PL/1 Program of GP-P—Ungrouped; (6) Source PL/1 Program of GP-P—Grouped.

An objective of the experimental design is to get (1) as free an expression as possible of the programmers' own observations and opinions of the results of the grouping processor (although these may be biased by adaptation to regular listings, etc.); and (2) more objective measures of performance, such as: success or failure in making a modification or finding an error; rate of work; and extent of use of computer and other resources.

In many quarters, it is an article of faith that the abilities of programmers differ tremendously. [Youngs' (1970) thesis does not fully support this faith.] Regardless of the extent of differences between people, the differences are a factor that was considered in deciding how to observe the programmers. This factor was handled by selecting a homogeneous group of programmers at UCI, and by similar personnel selection at the commercial facility.

For the FORTRAN grouping experiment, programmers were students at UCI, all of whom came to the same class with the same background of previous programming courses.

At the commercial facility, programmers were judged by their supervisors to be of essentially equal competence.

The experimental design allowed the use of objective results for overall evaluation of the idea of a processor to introduce conceptual groupings into a listing or other program display medium. When the results of this study show statistically significant benefits, then such benefits can be predicted with confidence for a more efficient and more sophisticated grouping system in conjunction with its corresponding compiler.

The subjective results were used primarily to identify specific features which did or did not contribute to the overall results. For example, if everyone (or almost everyone) agreed that a feature such as showing the nesting of DO-loops was (or was not) helpful, then one would suspect that it did (or did not) contribute positively to the overall results.

The experiments for Conceptual Groupings in FORTRAN consisted of a 3 X 2 analysis of variance type design. The two selected variables were:

(1) Extent of post-processor intervention. There were three extents: (1) No intervention, (2) use of the half of the grouping methods which we consider "best," and (3) use of all grouping methods.

(2) Quality of FORTRAN program structure and commentary. There were two quality levels: (1) "Good," in the sense of being clearly better than normal practice, but not conspicuously outstanding; (2) "Fair," but not atrocious.

The resulting six data cells then appear as illustrated in Figure 6.

		"Good"	"Fair"
Extent of Intervention	None		
	"Best 1/2"		
	All		

Figure 6
Experimental Design for Groupings
Experiments in FORTRAN

5.3 Execution of Method

In all, 64 programmers were utilized in the FORTRAN grouping experiment among the student population at the University of California, Irvine. The programmers were to make specific enhancements to a selected FORTRAN program. The experimental design required at least ten programmers to incorporate the program enhancements under the correlations of each cell. The independent variables to be measured were: (1) time taken to complete the task; and (2) degree of success.

The experimental design suggested that if statistical significance can be achieved, the data will obviously contribute to answers to a number of questions, including the following:

1. Does the groupings processor help maintenance of FORTRAN programs?

2. Do the groupings help less (or more) on "good" programs?

3. How good is our judgment of which groupings methods in FORTRAN are most helpful?

The experimentation involving PL/1 was conducted at a local industrial plant. (It was selected because PL/1 was being used there, and because the management agreed to let AMS conduct experimentation at the plant.) However, because of the small number of programmers the plant could make available to us, the experimental design was compressed from six cells (as shown in Fig. 6) to two rationale: Use, or non-use, of the PL/1 groupings program.

6. RESULTS; EFFECTS ON MAINTAINABILITY

6.1 Introductory Findings

6.1.1 Previous Data

Although the data obtained on conceptual groupings was by interview and manual experimentation on the previous CASMS and RICASMS efforts (Overton et al., 1971, 1973) two major facts emerged about conceptual groupings and their usage:

First, the groups that maintainers use are small in terms of the number of lines of code involved.

Second, a very small number of group types include a very high fraction of all observed groupings. These group types are for the most part simply defined; they should lend themselves easily to automated maintenance aids for clarifying programs to be maintained.

In terms of the sizes of explicit groupings, the findings were: The groupings were (1) Small, but (2) Skewed in distribution toward the larger sizes.

Furthermore the experimentation showed that the seven most frequently specified groups were:

(1) I/O: This group consists of input/output and closely related statements.

(2) DO: A group starting with a DO statement and ending with the last executable statement within the range of the loop.

(3) IF: A conditional statement primarily involved in an if statement.

(4) GO TO: Ending with an unconditional transfer.

(5) ASSIGN: A group of statements, mostly FORTRAN assignment statements.

(6) DEC: Consists of FORTRAN declaration statements.

(7) DESCRIP: Consists of COBOL description statements (data or file).

The clear implication of these two results is that the known characteristics of conceptual groupings should be incorporated in actual maintenance aids. These findings were taken into consideration in the design of the Conceptual Grouping experimentation using PL/1 and FORTRAN under the current effort.

6.1.2 Confirmatory Observations

While this was the first time that pilot programs had been developed to attempt to display Conceptual Groupings, they had been manually studied previously. Those studies produced specifications as to the kinds of statements which typically marked the beginnings and ends of groupings, and the kinds of statements they typically contained.

These findings--statistical markers in a sense--were the guides for the pilot programs developed here.

They were supplemented--in PL/1 particularly--by observations of what were called "marked-up" program listings. This phrase refers to a technique which was developed previously: Rather than having every programmer talk out loud, some were told to simply draw rough cups or brackets around groups of statements as they read and studied them. It was found that if the programmers did not have to write down any words--just mark lines--their markings roughly corresponded to their orally-expressed conceptual groupings.

The marked-up FORTRAN sheets collected here tended to confirm previous findings regarding the characteristics of conceptual groupings in FORTRAN.

Those in PL/1 led to the conclusion that programmers tend to treat PL/1 like the older languages, which they learned first, which they happen to be most familiar with. Therefore other languages, in addition to the marked-up listings, were used as guides to the development of the grouping program for PL/1. Included in the other languages was COBOL, of which a pilot study had been made.

After the programs had been developed, the primary use of them was, of course, in experimentation. However, to confirm the results of the experimentation, some programmer opinions were solicited.

The salient finding was that no one objected to the groupings display; and most felt that the grouping concept was helpful.

Suggestions included:

(1) The more frequently control is transferred to a given point in a program, the more emphasis should be placed on that point.

(2) Some information available in conventional documentation should be incorporated into the displays of groupings. For example, system subroutines (as distinguished from subroutines in the application program) should be identified.

(3) If the original programmer specifies the range of statements to which a comment applies, this information should be used to specify a grouping. This could be broken into subgroupings if necessary.

Because these suggestions arose late in the course of this project, it was not possible to implement them in the present pilot programs. If they were implemented as appropriate in the programs, and if further data on PL/1 groupings were obtained, such groupings processors should offer even more practical help to maintenance programmers.

6.2 Grouping Studies Using FORTRAN

This was the most extensive experiment carried out and involved 64 student programmers at the University of California at Irvine. Most were going to school part-time and were employed full-time in industry. Experimentation was incorporated into normal classroom activities by the professor, Dr. Peter Freeman. The assignment was to incorporate selected changes in a FORTRAN Program (see Appendix B) which had been grouped according to the experimental conditions outlined in paragraph 5.2 above. The time spent to incorporate the change was recorded. Then the professor reviewed the programmers' listings and program execution data and assigned an adequacy score. The "raw data" from the experimentation are presented in Table 1.

TABLE 1

RAW DATA CONCEPTUAL GROUPINGS STUDY
FOR FORTRAN

Experimental Condition*	Programmer	Adequacy Score	Time of Effort (min)
1	3	70	20
1	16	65	45
1	33	55	40
1	36	30	55
1	38	35	58
1	43	40	50
1	44	30	45
1	57	65	45
1	63	40	45
1	64	45	55
2	4	65	59
2	12	65	41
2	21	70	60
2	22	30	52
2	25	75	20
2	29	75	49
2	47	30	40
2	50	50	50
2	52	75	34
2	59	65	40
2	61	65	58
3	1	75	30
3	7	75	35
3	14	30	51
3	26	65	48
3	39	75	50
3	40	40	48

TABLE 1 (continued)

Experimental Condition*	Programmer	Adequacy Score	Time of Effort (min)
3	41	75	42
3	42	70	39
3	53	70	28
3	56	40	55
3	58	75	45
3	60	75	60
4	6	75	60
4	9	65	50
4	10	70	59
4	11	65	40
4	23	75	45
4	28	75	31
4	35	75	35
4	49	35	50
4	51	40	15
4	55	75	26
5	8	70	32
5	15	75	58
5	20	75	30
5	31	75	46
5	34	50	35
5	37	70	55
5	45	30	49
5	46	75	19
5	48	75	59
5	54	65	59
5	62	30	55
6	2	75	43
6	5	40	52
6	13	55	33
6	17	70	17
6	18	75	55
6	19	75	43
6	24	55	65
6	27	50	60
6	30	75	40
6	32	40	40

*Condition 1: Minimal commenting, no grouping
2: Minimal commenting, "best-half" grouping
3: Minimal commenting, full grouping
4: Full commenting, no grouping
5: Full commenting, "best-half" grouping
6: Full commenting, full grouping

The data are summarized in Table 2. The basic comparisons of interest are those among the six experimental conditions. (The statistically "combined conditions" are shown in the bottom row and right column of the table.) The meaningful comparisons are between the mean adequacy scores under different conditions. It is immediately seen that the worst (or lowest) mean is a product of minimal commenting and no help from the grouping program.

Significant improvement appears under two conditions: (1) when the "Best Half" grouping algorithms are applied to minimally-commented code, and (2) when the full set of procedures is applied to the same code.

(The word "significant" is used in its statistical sense as well as in its ordinary connotation. When minimal commenting is used, both the "Best Half" and the full grouping procedures give results which, according to the "t" test for the difference between means, differ significantly from the results with no grouping procedures. The t values are 1.867 for the "Best Half" and 2.401 for the full procedures; the "degrees of freedom" necessary to interpret these t's in standard statistical tables are, respectively, 19 and 20. According to the statistical tables, the probability of the results arising by chance is less than .05.)

Thus, in the realistic condition of poorly-commented FORTRAN, this prototype grouping program is shown to be an effective aid to the maintenance programmer.

TABLE 2

SUMMARY DATA
CONCEPTUAL GROUPINGS STUDY FOR FORTRAN

	No Grouping Procedures	"Best-Half" Grouping	Full Grouping Procedures	Combined Conditions

Minimal Commenting	M* = 47.5
	N* = 10.
	S* = 15.1
		M = 60.4
		N = 11.
		S = 16.6
			M = 63.8
			N = 12.
			S = 16.8
				M = 57.7
				N = 33.
				S = 17.2

"Full" Commenting	M = 65.0
	N = 10.
	S = 15.1
		M = 62.7
		N = 11.
		S = 17.8
			M = 61.0
			N = 10.
			S = 14.7
				M = 63.9
				N = 31.
				S = 15.5

Combined Conditions	M = 56.2
	N = 20.
	S = 17.2
		M = 61.6
		N = 22.
		S = 16.8
			M = 62.5
			N = 22.
			S = 15.6
				M = 60.2
				N = 64.
				S = 16.5

Notes: *M = Mean score of adequacy of maintenance effort.
N = Number of programmers.
S = Standard deviation of scores.

6.3 Grouping Studies Using PL/1

In brief, PL/1 results were not comparable to those with FORTRAN. They were encouraging, but further development of the PL/1 grouping concept, in relationship to other maintenance tools, is needed.

Experimentation with FORTRAN provided dramatic evidence that the Grouping Program significantly facilitated maintenance. Experimentation with PL/1 turned out to be anticlimactic. The PL/1 Grouping Program did not fail to be helpful, but the results were obscured by several factors: Intrinsic differences between PL/1 and FORTRAN and effects of these differences on programmers, different availability of compiler features, and the less thorough development of the PL/1 Grouping Program.

PL/1 study procedures and results are described and discussed below.

6.3.1 Procedures

There were about 2,100 statements in the PL/1 application program which was used to test the PL/1 Grouping Program. The application program was essentially a large report writer capable of presenting a large variety of labor cost data and other financial data in various formats. It was the subject of considerable maintenance work at the installation using it, which was the computer service department of the electronics and related divisions of a large, diversified corporation.

Enhancements and other modifications were actually being made in the applications program by department programmers who were not among the original developers of the program. Two sets of enhancements were used in the studies reported here. (Time did not permit more extensive experimentation.) As a result of this working situation, the studies were cast in a real-life environment and not in an artificially contrived situation.

The two sets of enhancements were made by two different two-man teams of maintenance programmers: a senior and a junior programmer formed each team. Their supervisors gave them the assignments and mentioned, without elaboration, that they would use a new kind of program listing in their work.

Unlike the FORTRAN procedure, comparable programmers were not given the same assignment with a different listing. Instead, the supervisors made the assignments with a view of past work which they thought was equivalent in difficulty, and with which they could compare the results with the listings from the PL/1 Grouping Program. The basic data, then, were simply the percentages of the man-hours needed to do the present assignments in comparison with past assignments (to the same people) which were thought to be equivalent.

It should be emphasized that these percentages were estimated, not by A.M.S. personnel and not by the working programmers themselves, but by the programmers' supervisors on the basis of time cards and other records and questioning of the programmers. The making of the estimates by the supervisors may have introduced a slight bias in favor of the normal procedures of the supervisors' department and against the experimental listing. Of course the extent of this bias, if any, is not known.

In contrast to the quantitative estimates sought from the supervisors, qualitative reactions were solicited from the programmers. The reactions were requested directly by A.M.S. personnel, and not by the programmers' supervisors.

In summary, these procedures were planned to produce two quantitative estimates and two sets of qualitative opinions.

6.3.2 Results

One assignment was estimated as a four man-day job on the basis of a presumably equivalent past assignment using ordinary listings. Using the experimental listing produced by the PL/1 Grouping Program, the assignment was completed in about ten per cent less time than this, that is, in a little over 3-1/2 man-days.

The second assignment was estimated, by the same procedure, as a nine man-day job. It was completed in a little over 7-1/2 man-days, for a fifteen per cent saving in time which might be attributed to the Grouping Program.

Great consistency was found among the subjective opinions of the four programmers. (There was no difference of opinion between the junior and the senior programmers.) Two, who had been using conventional listings, described the grouped output as "Great!" and "very, very helpful."

The reaction of the other two was more complicated. They had been using a "Checkout Compiler" as an option with IBM's "TSO" or Time-Sharing Option" with the department's IBM system. They tended to equate the Grouping Program with the Checkout Compiler, which was described as being "invaluable."

In their opinion, the Grouping Program treated commentary better than did the Checkout Compiler. But they criticized the Grouping Program for relying too much on GO TO's as signals of conceptual groups.

In summary, the results were consistently favorable but they were based on insufficient experimentation to be statistically significant and they may have been confounded by the effects of other tools to which two of the programmers were accustomed.

6.3.3 Discussion

Some minor observations indicated that the present version of the PL/1 Grouping Program should conform more closely to the characteristics of the PL/1 language. For example one programmer observed that the FORTRAN version helped show up a "trick" whereby a limitation of FORTRAN in the treatment of negative limits and subscripts could be avoided; he said that since PL/1 does not have this limitation, the trick would not be necessary and therefore would not have to be pointed out.

More basically, the reported over-reliance on GO TO's may indicate the program does not yet conform as well as it should to the greater degree of block structure in PL/1 as opposed to FORTRAN.

It was also a general opinion that programmers tend to use sub-sets of PL/1 which are comparable to whatever older language they happened to be using before they started on PL/1. Perhaps, in retrospect, more work should have been done on older languages before beginning PL/1.

Finally, future, practical versions of the PL/1 Grouping Program should be coordinated with other tools such as the Checkout Compiler, to complement as much as possible the features of the other tools.

7. IMPLICATIONS AND RECOMMENDATIONS

7.1 Developing Automated Guidelines

The methodology has now been evolved for defining, and validating through experimentation, computer software which will be of value in aiding the software maintenance process ("Guidelines"), and in measuring the degree of maintainability ("Metrics") of software under development. The Conceptual Grouping Guideline for FORTRAN and for PL/1 can be specified as a guideline for future AFSC software development. The prototype computer programs outlined in Appendices B and C can be converted into operational routines. (These could be incorporated as post-processors to their respective compilers, or they could be developed as modules of a computer-aided software metrics system.) Measures of conceptual grouping conformance can easily be specified which will evaluate software under development and give indications of the places where improvement could be made.

The Conceptual Groupings guideline has been virtually defined for FORTRAN, and most of the development for PL/1 is complete. The methodology should next be applied to the other major programming language in the AFSC inventory, COBOL.

The methodology developed is applicable to developing other maintainability guidelines and their corresponding metrics. A careful review should be conducted of the software maintainability guidelines that have been evolved to aid in solving the fundamental factors which inhibit a programmer from maintaining computer programs he did not develop. Selection should be made of these guidelines from ESD TR 72-121 (Overton et al., 1971) and ESD TR 73-125 (Overton et al., 1973) which have high potential for both enhancing software maintenance and for effective quantization. Of immediate attention are the DISTANCE guidelines and the NOTE guidelines. Objective measures need to be developed for each selected guideline. Then experiments would be designed to objectively evaluate the measures, experiments which are economic, and for which sufficient programmer populations can be selected to establish reasonable statistical significance to the results. In this manner a set of automated guidelines can be developed to both assist in and evaluate software maintainability performance.

7.2 Developing A Computer Aided Software Maintenance Metrics System

It appears feasible to begin the synthesis of the research and development efforts conducted in a Study of Fundamental Factors Underlying Software Maintenance Problems (ESD TR-72-121), and Research Toward Ways of Improving Software Maintenance (ESD TR-73-125) and this effort in Development of Computer Aided Software Maintenance into a Computer Aided Software Maintenance Metrics System.

The envisioned graphics system would encompass both computer assisted techniques of software maintenance evaluation while new systems are under development and of program analysis, change determination, and error discovery, for systems undergoing test and change.

The approach taken would be to develop a conceptual structure of the overall interactive graphics system, implementing selected maintainability guidelines in the form of metrics sub-program modules as their definition and quantization are established and verified. The graphics system as developed should be compatible with more than one potential user system of interest, and provide for handling FORTRAN, PL/1 and COBOL programming languages.

The design of the executive and control structure of the Metrics System should use a top-down structure. The design should begin with a global system flow chart and description, followed by detailed system decision descriptions for all the system functions.

Although it is called a "program" to emphasize its automatic nature, the contemplated Metrics Program is actually a system of programs. A possible, simplified, top-level flow chart of the overall system is shown in Figure 7 . The major factor behind the proposal of this particular kind of configuration is that it permits flexibility in the detailed design of the component Metrics Sub-programs. In addition to executive and control, other functions are combining the results of the Metrics Sub-programs, and input-output functions.

The Combining Module is important. Initially some simple scheme should be selected for calculating a weighted average (of the ratings of other sub-program maintainability) from the different ratings of the Metrics programs. At some future time, however, it would be desirable to develop an "intelligent" Combining function which would include the use of people's judgments (of program maintainability) as a criterion against which to optimize the weighting factors. In other words, it might be well to assume that people are still better than computers at deciding how hard it is to change programs, and have the computer system conform itself to the human standards.

By maintaining a top-down structure to the metrics executive system, and a modularized approach to the metrics sub-programs, a highly effective graphics system could be evolved over time. The system could be revised and improved to better assist the maintenance programmer as more techniques are identified and quantized.

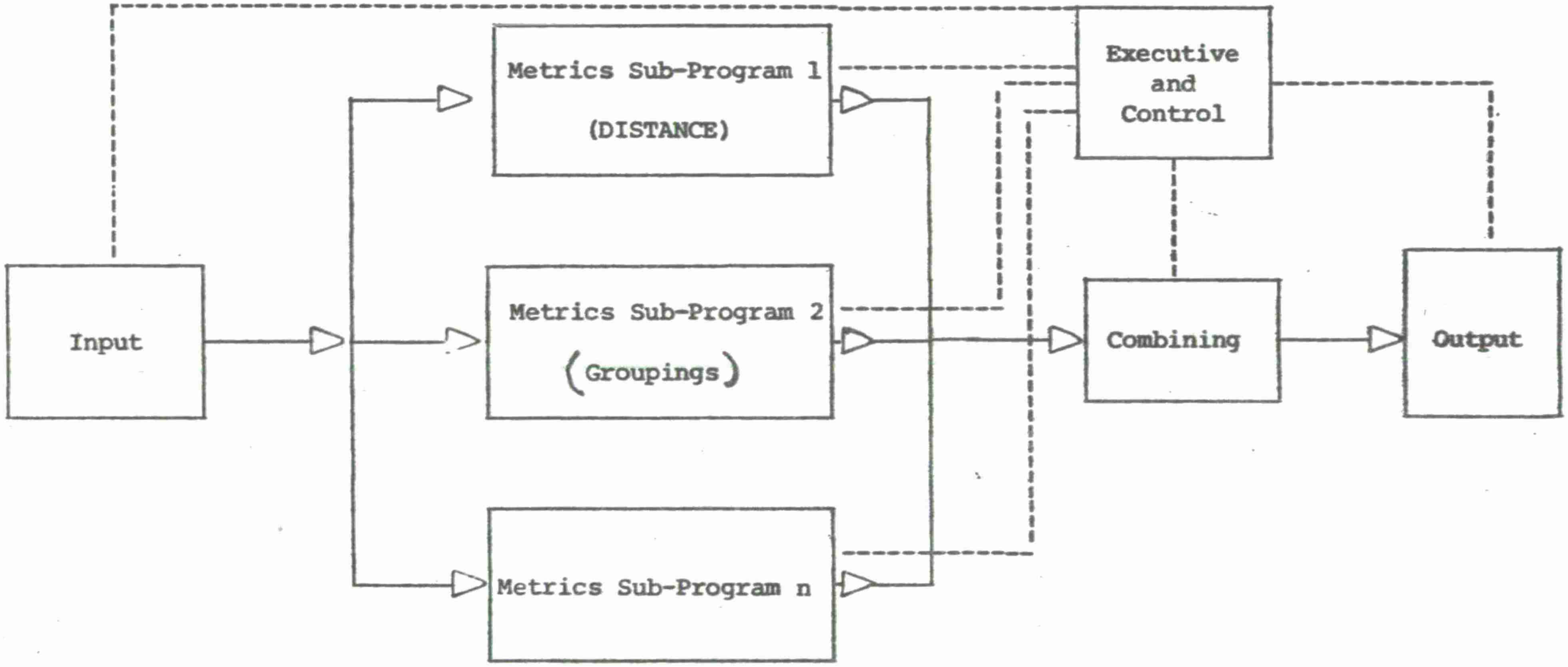


Figure 7. Proposed Top-level Flow Chart of Metrics System

7.3 Recommendations for Immediate Implementation

The principal purpose of the studies in conceptual groupings was the evaluation of such groupings as aids to maintenance; i.e., do they help the maintenance programmer, and can they be automatically displayed for his assistance? The answer, disregarding the qualifications and cautions that one would expect from a preliminary development project, was yes.

Accordingly, the basic recommendation is (to quote from an abstract of the study): "Pilot programs were developed. . . . Such programs merit wider usage."

The "merit" is economic. Beginning with an estimate of U.S. annual software cost of $10 billion (Boehm 1973), and (1) assuming that U.S.A.F. costs are 10% of that total; (2) 50% of the U.S.A.F. cost is applicable to software maintenance; (3) the groupings concept applies to only 20% of the software documentation; and (4) a minimum of 10% efficiency improvement is experienced; then, the total savings to the Air Force would exceed $10 million per year!

To be most profitable, the future work should be coordinated with available compiler tools (as discussed in Sec. 3.1.1). New groupings programs should also include the new features described in Sec. 6.1.2.

In addition to the continued development which the above basic recommendation implies, some recommendations can be made which can be implemented in short order, or by administrative fiat. These are listed below.

The listings include the sections of this report which present the backgrounds of the recommendations. Some references are also made to a following section (8.1, Alternative Perspectives of a Program) which overlaps both the graphic terminal arrangements and automated groupings studies.

(1) Conventional documentation of a program would probably be improved if programmers were required to use smaller documentation units. Such requirements or encouragements, by administrative directive, are recommended.

(1a) "Smaller" means, for most programs, and in rough, order-of-magnitude terms, that the average documentation unit should include closer to six than to sixty statements. (See Sec. 5.1.)

(1b) For programs of high intrinsic complexity (which scientific programs may be, in contrast to simple bookkeeping programs), the value of smaller functional units is even greater. (See Sec. 3.2.3.)

(2) Programmers should be encouraged to work in terms of small computational routines for performing one and only one transformation on one variable or subvariable (as contrasted with routines, which may be more efficient in computer usage, which utilize a statement to affect more than one variable). (See Sec. 8.1.4.2.)

(3) In software systems having a definite hierarchy, higher-level groups should involve only higher-level variables. (For example, total transportation cost is a higher-level variable than either loading costs or trucking costs; and total cost is a higher-level variable than total transportation cost.) (See Sec. 8.1.4.2.)

8. FUTURE RESEARCH RECOMMENDATIONS

8.1 Alternative Perspectives of a Program

Useful research is sometimes inspired by a new view of an old problem. This section puts forth the radical view that all programming is simulation, and that the problems of maintenance programming relate to the inadequacies of some simulations.

8.1.1 Programming as Simulation

A novel view of computer programs has been offered by Licklider (1973, p. 199):

> "Instead of thinking of a computer program as a procedure for solving a problem, we can think of it as a description of a process——as a description of how some system works. The system may be simple or complex, actual or merely imagined. In a computer program, one can describe it precisely and definitely. Then, when, the program is fed into a computer and the 'start' button is pushed, the description turns into a working model or simulation."

From this point of view, then, an inventory control program merely describes the work which a human clerk would do if he were fast enough; and routines for computer-aided design try to simulate human mathematician-helpers (again, at high speed).

In short, Licklider is saying that all programs are simulations. The present authors add two thoughts: (1) that the programmer's view of perspective of a program is also a simulation, and (2) that maintenance includes an attempt to bring the two kinds of simulations into some kind of congruence.

8.1.1.1 An Alternate Abstract

To illustrate the fact that different things can be seen in the same lines, the authors present an alternate abstract which was originally written only for diversion:

> "There were studies, notes and measures to find what helped and hindered people who modified computer programs. Environments were batch and time-share, using PL/1 and FORTRAN.

> People at time-sharing consoles were handicapped by scattered sources of the data that they needed; the summed effect of small distractions was a surprising loss of time.
>
> In batch, help was drawn from 'groupings' correlating lines and 'concepts.' Pilot programs were developed to automate display of 'groupings'; such programs merit wider usage."

This alternate abstract contains a rhythm, and people tend to notice it. But a text processing software system (which could accept the alternate abstract as input) could not "notice" the rhythm. Figuratively speaking, the computer lives in a world in which rhythm does not exist.

Computers respond to other things. For example, compilers like to count parentheses and report errors if open- and close-parentheses do not match.

These "things," such as the presence or absence of rhythm, or the matching or non-matching of parentheses, may loosely be called variables, attributes, or dimensions. Returning to the idea of simulations, any simulation may be described in terms of the dimensions along which it calculates. And, as will be discussed below, people and programs tend to work within the frameworks of different sets of dimensions.

8.1.1.2 Presence and Absence Effects

Once there was a payroll program which incorporated a defense against at least one form of sophisticated deceit by employees. The time cards, which the employee filled out and signed, were machine-readable, and they included a group of holes which told each employee's hourly rate of pay. The rate codes could easily be read by a person, too, if he knew the common, standard card format. So a sophisticated employee could have "doctored" his card to give himself a rate of pay of, say $99.99 per hour. (Also, he would be the only person to see his paychecks.)

As a defense against this possibility, the developers of the payroll program created a test for paycheck amount: If the amount was more than what seemed, to the systems analysts at the time, to be a reasonable maximum, the program refused to write the check.

Six years passed. There was personnel turnover in the business programming department. Then, when the company started laying off some professional employees, a "bug" appeared in the payroll program. It refused to write some of the employees' final paychecks. (Thus, incidentally, adding insult to injury on their lay-offs.)

After spending a significant amount of frantic people-time, plus spending some computer time, and suffering considerable embarrassment, the programming department tracked down the "bug." It was, of course, the old test for deceit in time cards . . . still working exactly the way the program said it should.

The program's problem was that (1) it took no account of--or failed to simulate--the effects of six years of inflation; and (2) it failed to simulate a world in which lay-offs took place--especially to well-paid people who might have accumulated up to four weeks' vacation time, to be added to their final paycheck. In this new, real world, the "reasonable" maximum of six years ago was no longer reasonable.

Conversely, to understand the apparent "bug," the current maintenance programmers had to mentally "simulate" a world in which employees might fraudulently alter their time cards.

In general, a maintenance programmer will be baffled by the workings of a system which simulates the presence of a dimension of which the programmer does not happen to think. Until he does think of it, he will be bewildered --like a creature of the mythical Flatland trying to imagine a three-dimensional universe (Abbott, 1884).

Similarly, the absence of a dimension in a program will cause a very common type of malfunction: The program does its simulation perfectly; it just does not simulate the world that the programmer has in mind.

8.1.2 Parameters of Simulations

The Licklider thesis (1973) is that a computer program is a description (or simulation) of a process which might occur in the real, outside world. The present authors have added that maintenance programmers do their work within the constraints of a mental "simulation" of the outside world. From this point of view, it is useful to list some basic parameters of simulations in general; and to note ways that they differ between people and programs.

Parameters include (1) the number of variables or dimensions involved in the simulation, (2) the size or capacity of the memory that it occupies, (3) the fineness of its categorization or digitizing, (4) its capacity for interpolation and extrapolation, and (5) its suceptibility to change.

8.1.2.1 Number of Variables

An engineer, wanting to predict the performance characteristics of a proposed airfoil, may request the use of a computer program. The program, in turn, may require data which give the density of the air at various altitudes. The density data may be viewed as a one-variable simulation of the world.

If one were concerned about the performance of a pilot rather than an airfoil, a simulation program would have to represent and combine the effects of many variables: air density, nutrition, work load, muscular strength, etc. To our knowledge, such a program does not exist.

More generally, computer programs are simulations of few-dimensional worlds. These are "exercised" in great detail, with well-known successes.

People, in contrast, are properly called upon to make plans which include many different aspects of reality. Efficient reorganization of a clerical office, for example, might require knowledge of the talents and personalities of the clerks, acquaintance with the user-level documentation of computer programs, familiarity with the various jobs the office has to do, awareness of the space and equipment requirements, some feeling for what the workers will and will not tolerate, and so on.

8.1.2.2 Memory Capacity

The simulation must be stored in something. One normally thinks of this something as being a computer memory. Certain simulations can require very large memories. By the standards of only a few years ago, computer memories available today are indeed large.

Turning to the "size" of the memory in which a person develops his "simulation," the situation is less clear. A remarkable fact, which has been elaborated elsewhere (Overton, 1961), is that estimates of the memory capacity of the human brain disagree by 15 orders of magnitude: by a factor of a trillion.

8.1.2.3 "Granularity"

The Air Force once supported Melpar in the building of an experimental maze-running machine which was attached to an immobile "learning network" (Carne, 1962). (Using company funds at another corporation, the authors developed a "learning machine" which was similar, but which was entirely non-physical: the maze and machine were both simulated in a computer.) As the Melpar machine buzzed around in a maze, the "learning network" built up, within itself, a sort of model of the maze. The model dealt primarily with the one variable of azimuth, or direction. This variable was split into four categories; that is, all turns were right angles. In other words, the system simulated an imaginary world in which 45-degree turns did not exist.

For a digital system, all variables must be digitized or categorized in some way. The fineness of digitization determines what might be called the "granularity" of the simulation or "picture" which the system produces.

Incidentally, there is a relationship between size of memory, and granularity. Obviously, finer categorization and smaller granularity call for the simulation of more points on each scale or variable; and, therefore, for more memory storage.

People, like the maze "learning network," tend to be broad-category systems. Suppose a person is asked to make absolute, non-comparative judgments of the absolute magnitudes of variables such as weights and noise levels; he tends to place them in only a few broad classifications: "very heavy," "medium," "fairly quiet," etc. Digital computers, in contrast, can usually record a measurement with all of the precision with which it can be measured.

Because people and computers use different "granularities," a bug developed in one of the programs for "learning." At one point, the program needed to choose at random among several possibilities. To make the choice, the programmer called in some digits which he considered insignificant and random: round-off error from a previous division. He had the program refer to these digits to make its choice (by a procedure like eliminating the first half of the possibilities if the first digit was odd, eliminating the next quarter if the next digit was odd, and so on.)

As the programmer should have known, however, the computer accumulated and erased round-off in a systematic manner; so the "random" choices were not really random; and the lack of randomicity was sufficient to make the program malfunction.

8.1.2.4 Power of Generalization

Any simulation must be based on what are often called "data points;" i.e., on known samples of relationships between variables. Typically the points as such are never seen by the programmer; instead, he uses an equation which "fits" the points. He programs the equations (and even business programs can be viewed as systems of discrete, logical equations) and then goes away happy.

(In some systems, the program itself performs operations which are the logical equivalent of fitting equations to the data.)

Some significant decisions are hidden behind the equations or their equivalent. The putative equations are essentially mechanisms for interpolating between, and extrapolating beyond, data points.

(1) How far shall extrapolation be allowed? At what point should one start losing confidence in the extrapolations?

(2) By what rules should interpolation be permitted, and with what degree of confidence?

(3) Is the user warned of extrapolations and interpolations which may not be accurate? (No is usually the answer.)

Less precisely speaking, these questions ask: What is the power of the program to generalize? And, how is this power reported to the user?

Programs, of course, possess whatever degree of generalization power that happens to be given to them. People, in contrast, have much more power and inclination to generalize than they may realize. A classic book (Bartlett, 1932) shows that throughout life people tend to substitute extrapolations and interpolations for "real" memories, and to confuse the two.

People's tendency to generalize--to "fill in the gaps" in a situation--helps account for at least one common type of bug: one in which the program makes a test for a narrow area, whereas the programmer has a larger area in mind . . . as when the programmer tells the program to go out of a loop at N = K, while he means N = K or greater. Then, if the incrementing process skips over N = K (as in counting by two, and skipping an integer), a bug develops: the program gets trapped in the loop.

8.1.2.5 Susceptibility to Change

From a detached and academic point of view, a "simulated outside world" should be capable of self-improvement; it should change, and become more realistic, as more evidence or data come in. In practice, of course, one normally does not want a computer program to start changing itself in any independent manner. (A payroll program would disturb the management if it decided to start giving raises to people.)

People do change their "simulations" as time and events transpire. Some people make changes more readily than others--they require less evidence before modifying their "view of reality." Somewhat surprisingly, there is no evidence that susceptibility to change (at least in some areas) has any correlation with differences in intelligence (Adorno et al, 1950). The present authors speculate, however, that there may be a correlation between programming ability and the variable of willingness to change one's view of "reality." Therefore, research in this area might contribute to more efficient programmer selection.

Returning to the computer program as a simulation, its susceptibility to change by the maintenance programmer, rather than by itself, should of course be high. That is simply a re-statement of the general goal of this project; and the detailed ways of reaching it would trace many of the steps of this and preceding projects.

8.1.3 The Embedding of Simulations

Occasionally managers complain that some programmers lack "common sense." (An example was a hospital administrator who cited a programmer whose program displayed most data in scientific notation. Hospital clerical helpers, who "didn't know an exponent from an expletive," were confused to find dates written as

$$2.5 \times 10^1 \text{ OCT } 1.969 \times 10^3.)$$

What the managers seem to mean by "common sense" is an accumulation of knowledge about the real world. Common sense is shown by an agreement of views: If the user and his environment are viewed in the same way by the programmer and the manager, then the manager says the programmer has "common sense."

As this rather cynical definition implies, measurement of common sense is not an absolute process: It depends on who is doing the measuring. Different people are not working with exactly the same simulations of reality. (From the user's point of view, the programmer and his manager may both lack common sense.)

The program (according to Licklider) simulates a little bit of the real world in which the user operates. But it is a very, very detailed simulation of that little bit. (It says, if it is a payroll program, that you eliminate leading zeroes before the most significant digit in writing a paycheck. And it specifies all necessary calculations before you get to that point.) On the other hand, the user's simulation of the simulation is usually quite attenuated. (He may only know that you put time cards in, and you get paychecks out.)

In general, then, there is a double spectrum of simulations:

(1) The program, as a simulation, suffers from more and more attenuation and over-simplification as it is represented by people farther and farther from the original programmer.

(2) The user's world, and his need for modification of the program, is similarly attenuated as one moves from the user to the programmer.

And at every point the program/simulation is embedded in some degree of simulation of the user's world and needs.

8.1.4 Implications for Maintenance

"Work on a program," especially in maintenance programming, includes thinking about where in the program to do the most detailed work. In the terminology of the previous section, work on a program involves manipulating a simulation which is embedded in another simulation: a simulation of a simulation of a small part of the procedures in the world, embedded in a more attenuated simulation of the user's world in which those procedures will be followed.

In these terms, the difficulties of the maintenance programmer stem from these causes:

(1) He is in the middle: He has to be sufficiently "in tune" with the user's simulations to incorporate them into his own--in other words, to understand the user. But then he has to turn to the ultimate simulation--the program --and show that he understands it sufficiently to change it.

In larger programming organizations, of course, there is usually some kind of middleman--called analyst, customer engineer, or some other nice title--between the programmer and the user. But the basic problem still remains: bringing two "simulations," which normally suffer from different degrees of attenuation, into agreement with each other.

(2) The maintenance programmer must find out what variables or dimensions he can ignore in the simulation/program on which he must work. He has to coordinate his modification (or mini-simulation) with all relevant variables, and with none of the irrelevant ones. And he lacks the time to "exercise" or trace through all of the aspects of the program which are not relevant to his modification.

8.1.4.1 Emphasis on Dimensions

Given these causes of difficulties, some suggestions can be made about possible ways of alleviating them. The first calls for emphasis on dimensions and variables rather than on calculations.

Initial program design should be organized in terms of "variables and dimensions." Broad input variables should be specified and then refined down to the level of the computing environment (in the COBOL sense of the word). Output variables should similarly be worked backwards to the computational results which generate them.

This practice would obviously make it easier for the maintenance programmer to reject the variables which were irrelevant to his modification. Thus he could "compress" the simulation/program into a new simulation which would be simpler but still contain the material he needed.

This style of program design is certainly not entirely new, although it has not been given the emphasis which we give it here. The major difference between this practice and current custom would lie in _when_ calculations were performed. Insofar as possible, all input variables would be read in, and stored in specified files, before any calculations were performed on any of these data. Also, computational routines would be designed to cope with blank files--by simply reporting them as blank, rather than by generating error signals. Dummy input variables might even be provided, so that the maintenance programmer could use them to add variables at a later date.

Similarly, partially-reduced data would be stored in derived or sub-variable files, and these would be filled (as needed) before further calculations were performed.

Re-combining of files would be done as output was approached.

This practice would resemble the currently-popular "top-down" programming, except that there the emphasis is on building a hierarchy of _calculation_. Here the emphasis would be on building a hierarchy of variables and variable _results_ of calculations. Here, also, instead of one pyramid of hierarchy, there would be a double pyramid between input and output.

As an example of this approach, consider a hypothetical program for use in the mining industry. Given assay results from samples for a prospective mining site, the program will do trade-off studies to help decide how much to concentrate the ore on-site (with more expense for greater concentration), versus shipping less concentrated ore (at greater shipping expense) to a larger and more efficient permanent mill.

One approach would be to proceed with the following steps: The programmer would generally flow chart the heart of the program: the algorithms for doing the trade-off calculations. Then he would work down from this heart, specifying the supporting algorithms, which would be tied in to the data input and output routines.

The approach advocated here would be to decide first on the basic dimensions of the simulations. One dimension might be a mineral; variables to be read along that dimension would include its concentration in the sample, its price, and possibly others. The provision would be made for derived variables; value of the mineral per ton of ore would be one derived variable, and provision would be made for others, through appropriate file reservations.

Complementing the "tree structure" of input files would be a corresponding "root structure" of output variables. The user's interests include investments, returns, and time. These and other variables might be sandwiched between basic output files such as cumulative return on investment, and more detailed intermediary files such as equipment depreciation schedules.

The heart of the program--the algorithms for doing the trade-off calculations--would come last. Its functions, in terms of comparing files and results, might be distributed among smaller modules than would be the case if it had been used as the point of departure.

If a bug were later found in (say) the calculation of copper values, or if new technology dictated a change in the costs of concentrations, it might be easy for the maintenance programmer, under this approach, to attenuate the overall simulation/program into only those portions which he needed to modify.

8.1.4.2 Transformation-Oriented Groups

A suggestion which logically follows, under the approach oriented here, is this:

> Conceptual groups should include--but not be limited to--computational routines for performing one and only one transformation on one variable or sub-variable.
>
> Higher-level groups or modules could then represent sets of related transformations of variables.

Referring to the mining example, one conceptual group might do nothing more than multiply copper prices by copper concentrations, to get the value of copper per ton of ore.

This is not entirely a new suggestion either. It is similar to the practice in a large maintenance project which sought "self-documentation" through the placement of commentary with "each little function." It also resembles the practice (which seemed very successful) at Copley Computer Systems, Inc.: Programs were built of "sections," each of which performed one and only one function. "Sections" were short, ranging from only two lines in length, through a mode of a small number, to a maximum of about 40 lines each.

It has previously been observed that there are different "representations" of the program: the program itself, its commentary, other programmer documentation, user documentation, etc. (The term "representation" comes from the study of ways in which technical articles can be abstracted. The article itself is one representation, its title is another; two different abstracts each written for a different purpose, could be two other representations.

It was also observed that there are interactions between the programmer, the program, and various representations of the program. Obviously, the interactions are best when the maintenance programmer can work with a representation which is tailored to his purposes.

The above two suggested practices would greatly help a programmer to create a representation of the program which was tailored to his purposes. He could do so by eliminating irrelevant variables and dimensions, and by concentrating on the "sections" whose functions he needed to change.

These practices represent a somewhat different philosophy than that which seems to be behind most of the calls for top-down programming. The difference in philosophy can be illustrated by analogies with maps and geometric figures.

In philosophy, most top-down advocates seem to envision program development as analogous to changing scale on a map. The top-level description is like a small map of the United States, on which many details are omitted. As one moves down to the coding level, details are added to the map, but its total configuration remains essentially unchanged.

The present approach views different representations of the program as analogous to projections of geometric figures into different universes. A sphere from a three-dimensional universe becomes a circle when it is projected into a two-dimensional universe; it becomes a line when projected into a one-dimensional universe. It does not just change scale; it changes form.

The maintenance programmer needs to work in as limited a universe as possible; modifications are made most efficiently when they can be made simply. For this reason, the philosophy of making it easy "to yank out dimensions" would also make it easier for the maintenance programmer to "zero in" on the places to make coding changes.

Finally, one should not give up other conceptual groups which programmers have found useful. Not all groups, which helped the programmers in these studies, were based on "each little function." The philosophy of "yanking out dimensions" is an approach to enhancing maintainability which does not rule out the use of other techniques and aids.

8.1.4.3 Function and Variable Displays

This approach also carries implications regarding the kind of displays that are likely to be most useful for maintenance programmers at graphics terminals.

Terminal operating systems should facilitate displays of (1) unitary functions which transform variables, and (2) the resulting, transformed variables.

In view of the apparent preference of many people for visual analogs, the transformed variables might, wherever feasible, be available in analog form.

In terms of this dimensional approach, operating systems like that just advocated would make it easier for programmers to simplify, or compress the picture, which they apparently like to do when they are puzzled.

More basically, it would also facilitate the making of the comparisons which maintenance programmers frequently have to make: comparison of one version of a short function with a later version of the same functional unit; and comparison of a functional unit with its effect on a dependent variable. The ease and speed of these comparisons has much to do with the speed at which maintenance programming progresses.

8.2 Recommended Research Projects

8.2.1 Computer Aided Software Maintenance Terminal Systems

In a study of the Fundamental Factors Underlying Software Maintenance Problems, ESD-TR-72-121 (Overton et al, 1971) two Research Plans were presented; one of higher priority, shorter range, moderate cost nature; and the other of lower priority, longer range and of greater cost nature. With the possible exception of the Terminal Arrangements Task all the other studies and experimentation to date outlined in ESD-TR-72-121, ESD-TR-73-125 and the current effort have been tasks emanating from the higher priority research plan. In the Terminal Arrangements Study, effort was begun to look at the overall environmental aspects of interaction between a programmer's sensory perception and computing system input/output equipment, including graphic terminals.

One of the research projects outlined in the longer range plan was the establishment of a Microcosmic Test Bed to more objectively study the use of the interactions among graphic terminals, structured documentation and hard copy. The Terminal Arrangements Study results and interpretations (in Secs. 3 and 4) indicate that there is need for greater integration of the sensory information inputs (and outputs) between the maintenance programmer and the graphics console. Of the independent variables identified (in Sec. 2.2.3.1), the experimentation was able to cover only those of program complexity and modularity. The Terminal Arrangements Study produced a wealth of possibilities for improvement in software maintenance utilizing graphic terminals.However, the results were based as much on field observation as on controlled, scientific experimentation. Therefore it is recommended that a Test Bed be established to confirm and make practical applications of these results, and to study the effects of other independent variables and possible maintenance aids in graphic terminal systems. It is suggested that the ARPA network be considered for the research test bed.

TASK 1. Software Maintenance Terminal System. Perform a study to establish optimum graphic terminal arrangements for computer-aided software maintenance. The effort would include:

(1) Select a test bed for experimentation. Make arrangements with existing selected time-sharing

network user (university, government, government sponsored) services and utilization of graphic terminal system as a test bed. Augment test bed as necessary to enable execution of objective experimentation.

(2) Develop Test Plan for experimentation with significant independent variables. Test Plan should be for at least two series of experiments, with a review and revision period between each series of tests.

(3) Execute initial experiments. Reword sensory data and experimental results. Perform analysis and present objective evaluation.

(4) Revise Test Plan and conduct next series of experiments. Reword sensory data and experimental results. Perform analysis.

(5) Write a report of results, specifying and recommending design guidelines for development of integrated computer Terminal Systems which are efficient for computer aided software maintenance.

8.2.2 Computer-Aided Software Maintenance Support Systems

8.2.2.1 Automated Maintainability Guideline Development

A methodology has been developed for defining useful guidelines, developing test procedures and conducting experiments to establish within reasonable doubt guidelines which can be automated and are effective in improving software maintainability. This methodology should now be applied to the study and verification of those guidelines suggested in ESD TR-72-121 (Overton et al, 1971) and ESD TR-73-125 (Overton et al 1973). The following tasks are, as a minimum recommended.

TASK 2. Conceptual Groupings for COBOL (GP-C). Perform a study of conceptual groupings in COBOL and their applications to the enhancement of the efficiency of maintenance programming. The effort would include:

(1) Select the subset of the COBOL language and determining the corresponding conceptual groupings.

(2) Plan observations of the use of groupings to obtain the most valid possible data.

(3) Design, Program and Debug a computer program which will be a post-processor to a COBOL compiler to create conceptual groupings of a source COBOL program.

(4) Observe the usage of groupings, modify compilers to distinguish such groupings, and observe and collect data on the value to maintenance programmers of such modifications, as a basis for later recommendations.

(5) Summarize and analyze the observations just described, to reach the most valid possible conclusions.

(6) Write a report of the results, clearly stating recommendations of value to maintenance programming. Deliver the prototype COBOL Conceptual Groupings Program (GP-C), including programming documentation and operating instructions.

TASK 3. Developing Automated Guidelines. Perform a study to derive automated procedures for, as a minimum, the DISTANCE and the NOTES guidelines. The effort would include:

(1) Review existing analyses of the fundamental factors which inhibit program maintenance and the corresponding maintainability guidelines, and select those guidelines the objective measures of which will significantly enhance computer aided software maintenance.

(2) Develop methods of measuring the effectiveness of at least the DISTANCE and NOTES guideline, and outline a program of experiments to verify the applicability of the derived measures.

(3) Execute the planned experiments in the approved test plan using the largest test sample size as practicable within the target costs involved.

(4) Based on an analysis of the data from the initial experiments, verify the effectiveness of the measures, modifying the measures and test procedures as necessary, and verify the revised measures of maintainability.

(5) Recommend those guidelines, the measures for which can be automated through computer assisted software.

(6) Prepare a report summarizing the research methodology used to develop automated guidelines for computer assisted software maintenance, together with the results of the test experimentation to verify and select the corresponding measures.

8.2.2.2 Automated Program Error Search

A recurrent theme throughout the interview and observation sessions of the Terminal Arrangements Study task was the maintenance programmer's need for better automated procedures to assist in the software maintenance effort. The potehtial use of decision theory as an aid in helping locate where an error is occurring or where a program change should be made was outlined in ESD TR-73-125 (Overton et al, 1973). It is recommended that the following effort be undertaken to develop and verify this effective methodology utilizing graphic terminals.

TASK 4. Computer Aided Software Error Search (CASES) System. Perform the design and development of a computer program error search system utilizing decision theory as a computer assisted software maintainability aid. The effort would include:

(1) Collect the background information necessary to plan, in terms of general flow charts, the proposed system. Review the relevant literature on different statistical techniques in decision theory, and on the classification of bugs. Develop a usable, realistic, expandable nomenclature of bugs. Design note- and history-taking routines. Design the statistical decision routines. Integrate the plans into a set of system flow charts. Insure that specifications, descriptions, and documentation are compatible with software maintainability and can make effective use of computer assisted software maintenance software systems.

(2) Develop and evaluate a preliminary System. Select a test bed in coordination with interested users, program a preliminary version of the system and evaluate and refine the design. Design and conduct experiments to bring out good and bad features, and data on these features, in the preliminary system. Design a refined and more generally usable version of the system in terms of flow charts.

(3) Implement a Prototype System. Perform a test implementation of the prototype system on a selected time-sharing network. Following the design in the flow charts from (2) and within the constraints of the time-sharing network, etc., reprogram the CASES system in prototype form. Install and in accordance with reasonable standards, debug the CASES system, making it available to interested users.

(4) Create a Final Report detailing the background and planning of the efficient debugging decision system and the lessons thus far learned in taking advantage of it to improve the efficiency of maintenance programming.

8.2.2.3 Computer Aided Software Maintenance Metrics System

It appears desirable at this stage to begin the design and development of an overall system structure of a computer assisted software maintenance system utilizing graphic terminals. The test bed for such a development should be some user time-sharing network of interest such as the ARPA or WIMMEX networks.

The envisioned graphics system will encompass both computer assisted techniques of software maintenance evaluation while new systems are under development and of program analysis, change determination, and error discovery, for systems undergoing test and change.

The approach taken is to develop a conceptual structure of the overall interactive graphics system, implementing selected maintainability guidelines in the form of metrics sub-program modules as their definition and quantization are established and verified. (See Tasks 2, 3, and 4.)

By maintaining a top-down structure to the metrics executive system, and a modularized approach to the metrics sub-programs, a highly effective graphics system can be evolved over time. The system can be revised and improved to better assist the maintenance programmer as more techniques are identified and quantized.

TASK 5. Develop Maintainability Metrics System. Perform the design and development of a Computer-Aided Software Maintenance Metrics System. The effort would include:

(1) Design the structure of a computer aided maintenance metrics system. The system shall be designed to operate on an inter-active basis, capable of being interfaced with a variety of graphics consoles and computer telecommunications systems. The program design shall be modular and expansible such that measures of maintainability guidelines can be added to the system as the measures are defined and successfully verified.

(2) Program and test the necessary portions of the Metrics System Program, and the Metrics Modules on an inter-active time-sharing system utilizing a graphics terminal.

(3) Test the prototype Metrics System, including the groupings, DISTANCE, NOTES, and other selected maintainability guideline sub-programs on an interactive time-sharing system utilizing a graphics terminal. The prototype Metrics System will be implemented in such a manner as to be transferable to an AFSC designated computer system.

(4) Prepare a report summarizing the prototype Metrics System design and implementation together with results of the test experimentations. Prepare program documentation and operating instructions for the prototype computer software.

8.2.3 Dimensional Approach to Maintainability

The maintenance programmer has to search for the exact parts of the program in which he should make changes. There are indications that the search is facilitated if the program can be viewed as a simulation and if the simulation can easily be compressed into various simpler forms by the removal of dimensions or variables. Also, explicit identification of dimensions may prevent the errors which are often caused by unwarranted implicit assumptions.

TASK 6. Dimensional Structures for Software Maintainability. The dimensional approach to program development would include the following tasks.

(1) Design computational routines able to operate on null or deleted dimensions. Develop other requirements for dimensional structures.

(2) Detail the approach sufficiently to specify its differences from current practices and to defend its input on program development costs.

(3) Write simple programs illustrating the approach.

(4) Conduct experiments using other programmers to modify the dimensional structured programs to evaluate the gains in maintainability as compared to current approaches such as modularity and structured programming.

(5) Create a final report outlining the preliminary findings as to benefits and efficiencies gained in software maintainability with the dimensional approach.

REFERENCES

Abbott, Edwin A. Flatland. New York: Dover, 1952. (Republication of revised edition of 1884.)

Abrams, Marshall D. "Remote Computing: the Administrative Side." Computer Decisions, October 1973.

Adorno, T.W., Frenkel-Brunswick, E., Levinson, D.J., and Sanford, R.N. The Authoritarian Personality. New York: Harpers, 1950.

Baker, F.T. "System Quality through Structured Programming." AFIPS, 1972.

Bartlett, F. Remembering. Cambridge: Cambridge University Press, 1932.

Bartlett, F.C. Remembering, A Study in Experimental and Social Psychology. Cambridge, England: University Press, 1962.

Bemer, R.W. "Manageable Software Engineering." Software Engineering, 1970. (Quoted in Computing Reviews, May 1971.)

Blee, M. "Modular Programming--Innovation or Common Sense?" Data Systems, February 1969.

Boehm, B.W. "Software and its Impact: A Quantitative Assessment." Datamation, May 1973.

Boies, S.J. User Behavior on an Interactive Computer System. Yorktown Heights, N.Y.: IBM Watson Research Center (AD 754 836), 1973.

Bousfield, W.A. and Sedgewick, C.H. "An Analysis of Sequences of Restricted Associative Responses." Journal of General Psychology, 1944, 30, 149-156.

Bower, G.H. "A Multicomponent Theory of the Memory Trace." In K.W. Spence and J.T. Spence (Eds.), The Psychology of Learning and Motivation. Vol. 1. New York: Academic Press, 1967.

Briggs, R. Visual Confusions with Aural Presentation, Paper at Convention of Amer. Psychol. Assn., Montreal, 1973.

Broadbent, D.E. Decision and Stress. London: Academic Press, 1971.

Brooks, R. Ruven. A Model of Code-Writing Behavior in Computer Programming. Dissertation, Carnegie-Mellon University, 1974.

Brown, R. and McNeill, D. "The 'Tip of the Tongue' Phenomenon." Journal of Verbal Learning and Verbal Behavior, 1966, 5, 325-337.

Carne, E.B. Self-organizing Models--Theory and Techniques. Falls Church, Va.: Melpar, 1962. (Air Force contracts AF 33(616)-7682 and AF33 (616)-2834.)

Coleman, M.L. "Genesis." Datamation, Nov. 1973.

Collins, A.m. and Quillian, M.R. "Retrieval Time from Semantic Memory." Journal of Verbal Learning and Verbal Behavior, 1969, 8, 240-247.

Fosdick, Lloyd D. "The Production of Better Mathematical Software." Communications of the ACM, July 1972.

Franklin, J. and Dean, E. "Computer-Aided Design with Interactive Graphics." S.I.D. Journal, 5-13, May-June 1974.

Hammerton, M. "Processing of Numbers and of Physical Magnitude." Perceptual and Motor Skills, Vol. 37 (1), 155-158, 1973.

Helson, H. and Steger, J.A. "On the Inhibitory Effects of the Second Stimulus Following the Primary Stimulus to React." Journal of Experimental Psychol., Vol. 64, 201-205, 1962.

Hill, L.W. "Data Communications Revolution Trends." S.I.D. Journal, 5-7, Nov.-Dec. 1973.

Jahns, D.W. "Operator Workload: What is It, and How Should It be Measured?" In Cross, K.D. and McGrath, J.J. (Eds.) Crew System Design. Santa Barbara, Calif.: Anacapa Sciences, 1973.

Kennedy, T.C.S. and Facey, P.V. Mini-Computer-Based Hospital Administration System. (Quoted in International Journal of Man-Machine Stories, April 1973.)

Kolers, P.A. "Translation and Bilingualism." in Miller, G.A. (Ed.), Communication, Language, and Meaning. New York: Basic Books, 1973.

Licklider, J.C.R. "Communication and Computers." In Miller, G.A. (Ed.), Communication, Language, and Meaning. New York: Basic Books, 1973.

Liskov, B.H. "A Design Methodology for Reliable Software Systems." AFIPS, 1972.

McGregor, Bob. "Program Maintenance." Data Processing, May-June 1973.

Madnick, Stuart E. and Alsop, Joseph W., II. "A Modular Approach to File System Design." AFIPS, 1969.

Mandler, G. "Association and Organization: Facts, Fancies, and Theories." In T.R. Dixon and D.L. Horton (Eds.), Verbal Behavior and General Behavior Theory. Englewood Cliffs, N.J.: Prentice-Hall, 1968.

Miller, G.A. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." Psychological Review, 1956, 63, 81-97.

Moyer, R.S. and Landauer, T.K. "Time Required for Judgments of Numerical Inequality." Nature, Vol. 215, 1519-1520, 1967.

Newman, J.R. Extensions of Human Capabilities Through Information Processing and Display Systems. System Development Corp., professional paper SP2560/000/00, 1966.

Nice, D.S. "Extraneous Lateral Stimulation and Distribution of Errors within Tachistoscopic Patterns." Perceptual and Motor Skills, Vol. 36 (3-2), 1234, 1973.

Overton, R.K. Thought and Action: A Physiological Approach. New York: Random House, 1959.

Overton, R.K. "Some Data and Comments on Brain and Computer Memory Capacities." Proceedings, San Diego Symposium on Biomedical Engineering, San Diego, 1961.

Overton, R.K., et al. Research Toward Ways of Improving Software Maintenance. ESD-TR-73-125. AFSC ESD, 1973.

Overton, R.K., et al. "A Study of Fundamental Factors Underlying Softwares Maintenance Problems: ESD TR-72-121 AFSC ESD, 1971.

Parnas, D.L. "On the Criteria to Be Used in Decomposing Systems into Modules." *Communications of the ACM*, December 1972.

Poincare, Henri. *The Foundations of Science*. Trans. George Bruce Halsted. Lancaster, Pa.: The Science Press, 1913.

Prokop, J.S. and Brooks, F.P., Jr. "Decision Making with Computer Graphics in an Inventory Control Environment." *AFIPS*, March 1971. (Quoted in *Computing Reviews*, April 1971.

Rhodes, John. "Tackle Software with Modular Programming." *Computer Decisions*, October 1973.

Sackman, H., Erikson, W.J., & Grant, E.E. "Exploratory Experimental Studies Comparing Online and Offline Programming Performance." *Communications of the ACM*, 1968, 11, 3-11.

Scott, Randall F. and Simmons, Dick B. "Programmer Productivity and The Delphi Technique." *Datamation*, May 1974.

Schuff, Fred and Schuff, Stephen J. "Mini-Modules Reduce Programming Effort." *Journal of Systems Management*, July 1973.

Semple, C.A. et al. *Analysis of Human Factors Data for Electronic Flight Display Systems*. Dayton, Ohio: A.F. Flight Dynamics Lab (AFFDL-TR-70-174), 1971.

Sime, M.E., Green, T.R.G., and Guest, D.J. "Psychological Evaluation of Two Conditional Constructions Used in Computer Languages." *Int. J. Man-Machine Studies*, 1973.

Simmons, Dick B. "The Art of Writing Large Programs." *IEEE Trans. Computers*, March/April 1972.

Simon, H.A. "How Big is a Chunk?" *Science*, Vol. 183, 482-488, 1974.

Sutherland, William R., Forgie, James W., and Morello, Marie V. "Graphics in Time-Sharing: A Summary of the TX-2 Experience." *AFIPS*, October 1969. (Quoted in *Computing Reviews*, November 1969.)

Tulving, E., and Pearlstone, Z. "Availability Versus Accessibility of Information in Memory for Words." *Journal of Verbal Learning and Verbal Behavior*, 1966, 5, 381-391.

Vrvels, D. and Schmidt, J.F. "Model for Effect of a Second Visual Stimulus Upon Reaction Time to the First." Perceptual and Motor Skills, Vol. 23, 323-328, 1966.

Walker, A.W. "An Interactive Graphical Debugging System." Computer Abstracts, February 1972.

Weinwurm, George F. (Ed.). On the Management of Computer Programming. Auerbach, Princeton, N.J., 1970. (Quoted in Computing Reviews, May 1971.)

Wirth, Niklaus. "Program Development by Stepwise Refinement." Communications of the ACM, April 1971.

Wortman, Paul M. and Greenberg, Leonard D. "Coding, Recoding, and Decoding of Hierarchical Information in Long-Term Memory." Journal of Verbal Learning and Verbal Behavior, 1971, 10, 234-243.

Youngs, E.A. Error-Proneness in Programming. Chapel Hill, N.C.: University of North Carolina dissertation, 1970.

Yntema, D.B. "Keeping Track of Several Things at Once." Human Factors, 5, 7-17, 7, 1963.

DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE

DCASM Final Report

APPENDIX A

LITERATURE EXTRACTS

(Prepared under Contract F19628-74-C-0061 by AMS, Inc.,
401 N. Harvard Avenue, Claremont, California 91711.)

APPENDIX A

LITERATURE EXTRACTS

This project included a literature review. More than two hundred potentially relevant reports and articles were reviewed, and approximately 75 were copied or filed. From those articles, some "quotable quotes" were selected. They are presented herein.

Since comments on the literature have been made, where appropriate, in the body of the Final Report, the literature extracts are presented here in a neutral mode, without any evaluative comments.

Abrams, Marshall D. "Remote Computing: the Administrative Side." Computer Decisions, October 1973.

> "All users will have questions. Their questions may be answered at many levels. In fact, when a question is first asked, the inquirer may not know what level of response he requires."
>
> " . . . it should be remembered that any exchange between two parties rapidly degrades to the level of understanding of the lesser party."
>
> " . . . computer centers have discovered that users cannot be permitted free access to systems programmers. The systems programmers could have all their time absorbed in answering trivial questions. But some users' questions will have to get through to the systems staff; this may be the staff's only feedback, or they may be the only people competent to answer the question."

Baker, F.T. "System Quality through Structured Programming." AFIPS, 1972.

> " . . . the Team operates in a highly disciplined fashion using principles of structured programming described by Dijkstra and formalized by Mills."
>
> "Although no statistics on number of errors or number of runs per module were kept, it was apparent

from a qualitative standpoint that both were significantly reduced when compared to similar systems on which team members had previously worked."

"The program library system used was also a major factor in improving quality. Ensuring that up-to-date versions of programs and data were always available reduced problems frequently encountered due to use of obsolete versions. For instance, when programmers were ready to use an interface, they could directly include the appropriate declarations into their code instead of writing their own version. When the interface changed, it was only necessary to recompile to incorporate a new version into all affected programs. In addition to reducing interface problems, the library system facilitated study of code to allow one programmer to adapt an approach used by another instead of re-creating it. Most importantly, it permitted the ready review and criticism of code by others as described above. As a side benefit, the availability of all this information in usable form reduced the need to get it verbally and thus further reduced errors due to distraction or interruption."

"This project has suggested two areas in which further work needs to be done. First, it may not always be possible to follow a strictly top-down approach in development of a large programming system. If a system organization, viewed as a tree structure, is narrow and tall, then a pure top-down approach may take too much elapsed time to be practical. Second, a more rigorous approach to code review needs to be developed. In retrospect, a number of the problems encountered in the Data Entry Edit Subsystem after delivery were of such a nature that they would probably have been caught earlier if all the code had been read."

Bemer, R.W. "Manageable Software Engineering." Software Engineering, 1970. (Quoted in Computing Reviews, May 1971.)

Subsection VIII.3.3 ("How Should Design and Implementation Be Partitioned?") provides opportunity for the author to state his feeling that the three phases of programming--system design, design of a routine, and coding--are best done by the same individual; this is a 180 degree reversal of the usual rule of thumb--have the three phases of programming done by separate individuals. His well-justified conclusion is that, except for small jobs, the entire project is best handled by the same person. Equating 'servicing' with program maintenance, for this job, the author rejects 'trainees' or 'experienced support personnel' in favor of the programmer who initially built or modified the software (called the 'originator'). 'If he considers it a trap, let him know that nothing but excellent and self-sustaining documentation will release him . . . .'"

Blee, M. "Modular Programming--Innovation or Common Sense?" Data Systems, February 1969.

This article discusses three techniques which differ considerably although all can be called "modular programming" in that they involve dividing a program into self-contained modules.

Brooks, R. Ruven. A Model of Code-Writing Behavior in Computer Programming. Dissertation, Carnegie-Mellon University, 1974.

"The theory consists of three basic processes; understanding, planning, and coding. While these processes are named in the order in which they will be discussed and in which they initially take place, in most situations they actually behave more like co-routines, with each processes calling on and being called by the others.

"Understanding

"A necessary prerequisite for a problem-solver to begin work is that he have some understanding of the problem. By this is meant that he has built up internal representations of the basic elements that the problem deals with and their properties, of the initial state of these elements, and of the desired final state, the goal. He must also have one or more operators which he can apply, appropriately, to transform the initial state.

"The information from which these internal representations and operators are built may come from a variety of sources. Some of the sources will be internal, such as knowledge of the problem-solving situation and general "world" knowledge. Others of these sources will be external, for example, the problem directions or the set of general directions if the problem is part of a larger problem set."

"Not surprisingly, evidence of an understanding processes, in the form of alternation between reading directions and reasoning about what they say, is also seen in records of behavior in programming problems."

"Planning

"The type of plan produced by this intermediate process is an algorithm for solving the programming problem; it consists of specifications of the way in which information from the real world is to be represented within the program and of the operations to be performed on these representations in order to achieve the desired effects of the program. These algorithms are used as schemas or templates to guide the writing of the actual code."

"The content of a plan is probably independent of the language in which the program is to be written. The basis for this assertion is partly a subjective one, the introspective reports of many programmers that they are able to think about solutions to problems without knowing the language that the problem will be written in. Reinforcing the subjective evidence is the informal observation that, having written a program in one language, it is easier to write it again in a second language, provided that both languages have similar operators and data types; if the experiment is arranged in such a way that a direct, language-to-language recoding is impossible, what must be carried over between the two situations is the algorith--i.e., the plan.

"While the content of a plan is independent of the programming language it will eventually be implemented in, the choice of a particular plan is clearly made with a specific programming language in mind. Programmers using FORTRAN do not usually select plans which involve list structures, nor do LISP programmers customarily set up their programs to use fixed-format, record input. If the opportunity exists, plans will be selected which are compatible with the language in which the program is to be written."

"Planning does not take place as a single operation; instead, an iteration occurs in which each plan that is created serves as input for the next cycles. Each cycle refines the plan and makes it more detailed. The terminating condition for the iteration is that some--reasonably large--part of the plan is sufficiently detailed so that the programmer feels that he knows how to translate it into code; at that point the final process in the writing of programs, coding, takes over. The coding process operates on a piece or part of a plan until either code is produced or some criterion is met which causes the coding process to report failure; when failure occurs information is passed back to the planning process which again attempts to produce a codeable plan."

"In many, if not most, programming problems, planning takes place extremely rapidly with little evidence of any kind of problem-solving activity. This suggests that what takes place is basically a match between characteristics of the current problem and the invoking requirements of a stored plan; the same mechanism might also account for cases which require a simple piecing together of parts of plans. In turn, this recognition mechanism also implies the existence of mechanisms for extracting characteristics from current problems and mechanisms for abstracting plans from solved problems."

"While the recognition system will probably take care of the overwhelming majority of cases, still other mechanisms will be necessary for the remaining cases in which a stored plan could not be used. These might be divided into two broad, general classes: those which use programming knowledge and those which use knowledge from the real-world problem domain for which the program is being written. In the former are included patching and re-arranging existing plans; generalizing from examples; and the use of diagrams or drawings. In the latter are included all those situations in which the programmer goes outside the programming domain and uses knowledge about the intended use of the program, relationships among the data, etc. to solve the problem; an example might be use of knowledge about a company's accounting policies to come up with a plan for writing a payroll program."

"Coding

"The third of three processes in the theory is coding. For human programmers, the basic cycle for the generation of code consists of using the plan to select and write a piece of code, assigning an effect or action to the code that has been written, and comparing the effect or action to the stipulations of the plan. The results of this comparison are used to select and write more code or to change the code that has been written; in turn, an effect is assigned to this new code which is compared to the plan. This cycle continues until the cumulative effect of the code meets the requirements of the plan or until some condition, such as effort expenditure, is met which indicates that the piece of plan is not codeable.

"The effects that are assigned to code are based on the differentiations among the data that the program must actually make in order to accomplish its purpose. Consider as an example a program for printing all the odd numbers in a set of integers; the program must differentiate between odd and even numbers in order to perform this task. An effect that could be assigned to a line of code in this program might be 'if the number is odd, this branches to statement 50,' a statement which uses the information about the odd-even distinction. The cumulative result of assigning this kind of effect to each line in a whole segment of code is to execute the code with symbols such as "odd number" replacing the real data; hence the whole process has been named "symbolic execution."

"After the basic cycle has generated a sufficient amount of code, the entire piece of code may be symbolically executed several times more. This may take place for one of two reasons. The first is to check the code that has been written to insure that there are no inconsistancies between its actual effects and the desired ones. The second reason is that there is no look-ahead in the basic generation to insure that all necessary prerequisites are met for using certain code structures before they are actually invoked; thus, required initializations and declarations are often omitted when code is first generated. A symbolic execution of an entire section of code often permits these omissions to be detected.

> "The symbolic execution cycle is not, of course, always successful at generating code giving a correct effect. When erroneous code is generated, there is no back-up to a correct state, as would take place in a tree-search problem solving process. Instead, the information in the effect of the wrong code and in the plan are used to attempt to correct the difficulty, often by adding additional code to fulfill unmet pre-conditions or by minor modifications in the code that has been written. In most cases, these corrections are successful; when they do fail to achieve correct code, the planning process may again be invoked to create a new plan or piece of plan which can be coded successfully. In turn, this may, in a few rare occasions, even result in a return to the understanding process. This means that as far as the progression from general plan to specific solution goes, programming has both top-down and bottom-up phases."

Fosdick, Lloyd D. "The Production of Better Mathematical Software." Communications of the ACM, July 1972.

> "There are simple, obvious things for programs written in the standard languages which would improve their portability. One is to put all machine-dependent parameters in one place, identify them as such, and give a prescription for changing them if the machine environment changes. Programs frequently have parameters which control storage allocation, execution time, and accuracy. Again these should be brought together, identified, and prescriptions given for changing them, which might help a user willing to sacrifice one for the other, say speed for accuracy."

Kennedy, T.C.S. and Facey, P.V. Mini-Computer-Based Hospital Administration System. (Quoted in International Journal of Man-Machine Stories, April 1973).

> "The first criterion for successful interactive use of a system is that it should be unnecessary for the user to refer to coding books or lists for command sequences or data entry. The user may be prompted in the case of incorrect entry. Commands should be simple in format and command verbs should be self explanatory. The most satisfactory data entry procedure has been shown to be a question and answer sequence since a positive request for data is given which reduces the possibility of omission.

Each entry may be validated as it is made allowing immediate correction in the case of error. It is possible, with this type of system, for a totally naive user to perform satisfactorily with the simple instruction that he must terminate any entry with the carriage return key before the computer 'understands' it.

"Secondly, the computer or terminal should not seem to take command. The user must maintain or appear to maintain complete control of the system."

"If these criteria are met, the man-machine interaction remains, on the surface, a simple, flexible procedure which allows a fast and efficient use of the computer. However, it calls for complex programs and a language which possesses powerful string handling facilities."

Liskov, B.H. "A Design Methodology for Reliable Software Systems." AFIPS, 1972.

"Levels of abstraction were first defined by Dijkstra. They provide a conceptual framework for achieving a clear and logical design for a system. The entire system is conceived as a hierarchy of levels, the lowest levels being those closest to the machine. Each level supports an important abstraction; for example, one level might support segments (named virtual memories), while another (higher) level could support files which consist of several segments connected together."

"Levels of abstraction, which will constitute the partitions of the system, are accompanied by rules governing some of the connections between them. There are two important rules governing levels of abstraction. The first concerns resources (I/O devices, data): each level has resources which it owns exclusively and which other levels are not permitted to access. The second involves the hierarchy: lower levels are not aware of the existence of higher levels and therefore may not refer to them in any way. Higher levels may appeal to the (external) functions of lower levels to perform tasks; they may also appeal to them to obtain information contained in the resources of the lower levels."

"Structured programming is obviously applicable to system implementation. We do not believe that by

> itself it constitutes a sufficient basis for system design; rather we believe that system design should be based on identification of levels of abstraction. Levels of abstraction provide the framework around which and within which structured programming can take place. Structured programming is compatible with levels of abstraction because it provides a comfortable environment in which to deal with abstractions. Each structured program component is written in terms of the names of lower-level components; these names, in effect, constitute a vocabulary of abstractions."
>
> "It is not clear exactly how early structured programming of the system should begin. Obviously, whenever the urge is felt to draw a flowchart, a structured program should be written instead."

McGregor, Bob. "Program Maintenance." Data Processing, May-June 1973.

> "New Systems development, in my opinion, cannot serve as a justification for lack of maintenance. Effective maintenance creates user goodwill. It gains user acceptance and assistance. It assists the user to perform more effectively."
>
> "On one hand his staff is interested in developing projects while goodwill is to be gained through satisfying the immediate needs of the user. The data processing manager is constantly faced with the problem of rotating staff from development work to maintenance work, dressing up maintenance work to look like it is something else and, in general, paying a very high cost for maintenance control."
>
> "I propose a different solution to this problem--the use of consultant programmers."
>
> "The first thing to consider is how such a concept could be put into practical operation. First, a position such as maintenance manager must be created. It must be filled by an in-house employee who reports directly to the dp manager."
>
> "The appointment of the maintenance manager gives the organisation a vehicle for developing individual skills at a prestigious level. Maintenance programming or 'fireman' work is an art unto itself, and requires special skills and talents. Rotating more senior staff members through this position

> will enable them to acquire these skills without feeling they are working beneath their capability."

Madnick, Stuart E. and Alsop, Joseph W., II. "A Modular Approach to File System Design." AFIPS, 1969.

> "The notions of 'levels of abstraction' or 'hierarchical modularity' can best be presented briefly by an example. Consider an aeronautical engineer using a matrix inversion package to solve space flight problems. At his level of abstraction, the computer is viewed as a matrix inverter that accepts the matrix and control information as input and provides the inverted matrix as output. The application programmer who wrote the matrix inversion package need not have had any knowledge of its intended usage (superior levels of abstraction). He might view the computer as a 'FORTRAN machine,' for example, at his level of abstraction. He need not have any specific knowledge of the internal operation of the FORTRAN compiler implementer operates at a different (lower) level of abstraction. In the above example the interaction between the 3 levels of abstraction is static since after the matrix inversion program is completed, the engineer need not interact, even indirectly, with the applications programmer or compiler implementer. In the form of hierarchical modularity used in the file system design model, the multi-level interaction is continual and basic to the file system operation."

Parnas, D.L. "On the Criteria to Be Used in Decomposing Systems into Modules." Communications of the ACM, December 1972.

> "Usually nothing is said about the criteria to be used in dividing the system into modules. This paper will discuss that issue . . ."

> "Below are several partial system descriptions called modularizations. In this context 'module' is considered to be a responsibility assignment rather than a sub-program."

> "This is a modularization in the sense meant by all proponents of modular programming. The system is divided into a number of modules with well-defined interfaces; each one is small enough and simple enough to be thoroughly understood and well

programmed. Experiments on a small scale indicate that this is approximately the decomposition which would be proposed by most programmers for the task specified."

"In the first decomposition the criterion used was to make each major step in the processing a module. One might say that to get the first decomposition one makes a flowchart. This is the most common approach to decomposition or modularization. It is an outgrowth of all programmer training which teaches us that we should begin with a rough flowchart and move from there to a detailed implementation."

"The second decomposition was made using 'information hiding' as a criterion. The modules no longer correspond to steps in the processing."

"In addition to the general criteria that each module hides some design decision from the rest of the system, we can mention some specific examples of decompositions which seem advisable.

1. A data structure, its internal linkings, accessing procedures and modifying procedures are part of a single module. They are not shared by many modules as is conventionally done."

"2. The sequence of instructions necessary to call a given routine and the routine itself are part of the same module. This rule was not relevant in the Fortran systems used for experimentation but it becomes essential for systems constructed in an assembly language."

"3. The formats of control blocks used in queues in operating systems and similar programs must be hidden within a 'control block module.' It is conventional to make such formats the interfaces between various modules. Because design evolution forces frequent changes on control block formats such a decision often proves extremely costly."

"4. Character codes, alphabetic orderings, and similar data should be hidden in a module for greatest flexibility.

"5. The sequence in which certain items will be processed should (as far as practical) be hidden within a single module. Various changes ranging from equipment additions to unavailability of

> certain resources in an operating system make sequencing extremely variable."
>
> "In discussions of system structure it is easy to confuse the benefits of a good decomposition with those of a hierarchical structure. We have a hierarchical structure if a certain relation may be defined between the modules or programs and that relation is a partial ordering."
>
> " . . . we must conclude that hierarchical structure and 'clean' decomposition are two desirable but independent properties of a system structure."
>
> "We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing."

Prokop, J.S. and Brooks, F.P., Jr. "Decision Making with Computer Graphics in an Inventory Control Environment." AFIPS, March 1971. (Quoted in Computing Reviews, April 1971.

> "This paper describes a well-conceived experiment in which 18 people participated. Near the end of 20 hours of instruction in advanced inventory control techniques, each participant was given statistics, resulting from two different simulation runs--for example, percent availability of stock, number of purchase orders generated, cost of sales, and total dollar investment in inventory. These runs contained 12 policies governing 34 items for 24 periods. At the end of each period, each participant ranked the policies in order of desirability.
>
> "In one run, all information appeared as hard-copy tabular listings. For the other run, subsets of the listing could be displayed via the Programmed Function Keyboard of an IBM 2250.
>
> "The results support the conclusion that better decisions can be made earlier and faster using displays instead of printout."

Rhodes, John. "Tackle Software with Modular Programming." Computer Decisions, October 1973.

> "Maintenance is one of the most expensive activities within any programming department: The original programming costs for a project often become small in relation to the aggregate maintenance costs after a few years of operation.
>
> "When planning any change, a programmer must understand the program to be changed; decide where and how to make the change; and check that the change will not produce unwanted results. Having made the alterations, he must retest and update the documentation. Modular programming can make these tasks simpler.
>
> "Since programs are subdivided, the programmer can easily identify the module to be amended, even if he did not write the original program. If carefully considered standards have been followed, alteration of the internal operation of the module will not cause undesired results. Retesting is necessary only for the module altered. And if documentation is also modular, changes to it are also easy."

Schuff, Fred and Schuff, Stephen J. "Mini-Modules Reduce Programming Effort." Journal of Systems Management, July 1973.

> "There is a growing interest in modular programming. If applied in a logical and organized manner, it may be quite a boon to the development of programming code."
>
> "This concept has proven very beneficial to the planning and development of large systems, but its initial success has kept the principle from being carried the next step further. If a large system can be modularized to enhance development, why can't these modules be 'mini-modularized' to enhance their programmability?"
>
> "Mini-modularization can be extended into several areas of programming tasks: standard calculation techniques, standard input/output (I/O) modules, and special utility functions."
>
> "The special utility function mini-modules are most valuable to the high level languages where there is usually no facility to perform the function no

matter how clever the programmer may be. Most of these routines are coded in machine language and employ features that are only available at this level. These functions may be thought of a frills, but after they become part of the system, they often become tremendous savers of time and effort. Examples are routines to intercept program failures (abends/interrupts) and allow corrective action by the application program; routines to provide information such as job name, data set names, CPU times, etc.

"A final consideration must be given to the fact that the long range cost of maintenance or upgrading a system at a later date is greatly reduced. Maintenance is limited to the code that is external to the mini-modules, due to their proven reliability and the search required to detect a problem is reduced significantly. The same is true for upgrading the system; there is less external code to be concerned with or, in the case where a mini-module is to be revised, the mini-module is upgraded and then replaced in all appropriate systems without affecting any other code in the system."

"As each new mini-module is developed, the base for building new systems grows larger and larger. This allows more of the time spent on each project to be allocated to other phases of the project or the total allocation can be reduced. The result can only be a reduced overall cost for the system in total."

Scott, Randall F. and Simmons, Dick B. "Programmer Productivity and the Delphi Technique," Datamation, May 1974.

"A review of programming management literature shows much commentary but very little research on programmer productivity."

"Simulation offers hope as a method of obtaining insight into programmer productivity; only recently has it been used to study computer programming. Of course, the use of simulation implies a knowledge of the active project variables and their interrelationships.

"Before beginning any programmer productivity research, it would be helpful to have a conference with some practicing managers and project management experts, although this would be both difficult to arrange and expensive. Even better results can be obtained using an inexpensive iterative method called Delphi."

"The members of a typical Delphi panel never meet each other. This eliminates the possibility of a small vocal minority swaying the responses of other members. Reputations are neutralized by the anonymous feature of the survey."

"The statistical group response feature provides for recognition that the exact answer is unknown and the value of all final responses should be maintained. This type of response helps reduce group pressure."

"Instead of future events, the Delphi statements were defined as variables. The panel members were asked to correlate each variable with programmer productivity, which was defined as implemented object instructions generated per unit of time. The panel members were asked to correlate, on a scale from minus seven to plus seven, the effect on programmer productivity of increasing the magnitude of each variable."

"Table 4 includes 22 variables on which there was a consensus after round two."

Variable	Median
"Quality of external documentation	6
Programming language	5
Availability of programming tools	5
Programmer experience in data processing	5
Programmer experience in functional area	5
Effect of project communications	5
Independent modules for task assignment	4
Well-defined programming practices	4
Central hardware	3
Quality of internal documentation	3
Personnel turnover rate	-3
Complexity of operating-system/ programmer interface	-3
Customer adp experience	3
Appropriate documentation standards	3

Table 4. Round 2 consensus

Table 4 (Cont'd.)

Variable	Median
"Availability of documentation aids	3
Peripheral hardware	2
Use of structured programming	2
Programmer participation in setting goals	2
Complexity of application	1
Number of installations	0
Shop environment (open or closed shop)	0
Number of unconditional transfer statements in the source program	0"

"These results clearly point up the importance that programming project managers place on providing the working programmer with a well-documented, thoroughly defined, independent task. Experienced programmers working in high level languages were also considered very important. Environmental factors such as hardware and operating system complexity, and open or closed programming shop did not receive high ratings.

"Another revealing result was variable 31 (number of unconditional transfer statements in the source program). There was a consensus since the range was one; but the median was zero. From this result it is obvious that this panel of experts does not feel the controversy on the importance of unconditional transfer statements ("GO TO" controversy) is worthwhile in its effect on programmer productivity."

Sime, M.E., Green, T.R.G., and Guest, D.J. "Psychological Evaluation of Two Conditional Constructions Used in Computer Languages." Int. J. Man-Machine Studies, 1973.

"There is a need for empirical evaluation of programming languages for unskilled users, but it is more effective to compare specific features common to many languages than to compare complete languages. This can be done by devising micro-languages stressing the feature of interest, together with a suitable subject matter for the programs. To illustrate the power of this approach two conditional constructions are compared: a nestable construction, like that of Algol 60, and a branch-to-label construction, as used in many simpler languages. The former is easier for unskilled subjects."

"The results of the present study indicate unequivocally that the NEST micro-language was easier for our subjects than JUMP. Reservations must still be made, however, when interpreting these results. For the complete beginner no separation can be made between ease of learning a language and ease of using it, yet for many practical purposes one might wish to trade off one for the other. Again, we deliberately avoided a source of severe difficulty in many programming languages, the expression of the scope of the conditional expression."

"Finally, it may be worth a small amount of speculation on the question of embedding. It was clear that the NEST group had greater difficulty with the harder problems. If, as seems likely, this is due to the increase in depth of embedding, the question is whether syntactic devices to reduce the depth of embedding would make a significant contribution. Such devices include well-tried methods, like the Boolean operators _and_, _or_, etc., which in the problems used would reduce the degree of embedding to zero and presumably make the task much easier."

Simmons, Dick B. "The Art of Writing Large Programs." _IEEE Trans. Computers_, March/April 1972.

"When writing small programs, one can use many unwise practices which have little effect on whether a program meets its design objectives as long as the program works. But, when writing large programs, poor program writing techniques can increase development time and cost and can cause maintenance difficulties after development."

"Many independent workers have come up with what could be considered facts that we must live with while developing large programming systems 1, 2, 3, 7. Some of these facts are:

"_Programmer Turnover_ - Anyone who has been involved in writing large programs has observed personnel turnover problems. Corbato has said that when planning a long term programming project, one should assume there will be roughly a 20% per year personnel turnover. Though industry does not like to advertise personnel turnover problems, a figure of 20% is also representative for many of the large programming projects developed in industry."

"Hardware/Software Turnover - . . . During program development when larger and faster disks, new types of remote terminals or even faster and larger processors become available, these are usually integrated into the system. Though computation centers try to minimize the effect on users during upgrading of systems, often the user is without a system for a considerable amount of time. The same is true whenever a new software system is introduced into a computation center. Often a major software system cut-over causes system outages."

"Major Ideas Incorporated Late - Major ideas to be incorporated in a program often originate after the program is written and nearly debugged."

"Program Never Debugged - No major program will ever be completely debugged. Throughout the life of any major program, bugs develop and have to be corrected."

"Program Maintenance - Every major program must be revised, updated or otherwise maintained. The program must be maintained by people other than the ones who originally wrote the program."

"Though there may be other important factors not included in the above, the list does contain salient facts that must be contended with during development of any large programming system."

"Any large program should be partitioned into modular blocks. Each block should be as self-contained as possible. The number of programmers working on the same module should be kept as small as possible. In many instances it has been observed that no more than ten people can be employed usefully in developing a single program module. A better limit would be six which is the largest number of people that should be under a single supervisor. A program should be partitioned so that interconnections between blocks are minimized."

"One way of monitoring is the buddy system where each program must be completely understood by the original programmer and at least one other programmer. During the writing and debugging phases each programmer continually interfaces with a buddy who is able to understand his program. The buddy system falls down when two inexperienced programmers are paired together."

"A design review committee made up of experienced programmers is another technique for monitoring program development."

"Both the buddy system and the design review committee divert manpower from the main task of developing programs. Though in the long run the manpower is well invested, it would be desirable to develop an automatic technique for monitoring program development. Automatic monitoring of program documentation is possible. For example, when flowcharts are produced directly from program listings, a high quality flowchart of proper detail can be produced only from a listing that has been properly commented."

"Proper grouping of program statements can greatly add to the readability of a program. For example, a program should not be designed so the control oscillates around a large area of statements thus requiring programmers to flip pages back and forth while trying to read the program listing. Branches of a conditional transfer should be placed close together if they eventually come back together in the main line of code. Programs should be properly broken into standard sections to make them easier to understand."

"The following documentation should exist in some form for programs:

"Program Description - As a first step in program documentation, the programmer's comments can serve as a program description. This description can be updated and modified."

"Program Listing - A program listing as we think of it today is the main item used by a programmer to create a program and to understand someone else's program. Comments are necessary whether a program is written in assembler language or in a high level language. A program should not be over-commented, but comments should be placed throughout programs to explain anything that might be unclear to someone reading the program."

"Comments placed within a program should state the purpose of a program sequence rather than describe the operation of program statements.

"Program comments can be classified as heading comments and explanatory comments. Heading comments should appear at the beginning of any major program section such as a program subroutine. They should explain the function of the program section, define inputs to and outputs from the section, etc. Explanatory comments are normally attached to program statements or immediately before them. These comments should explain what each section of a program does and should only explain what an instruction is doing when the function of the instruction is ambiguous."

"Data Layout - The data layout section of a program normally consists of data definition statements written in a one-dimensional syntax. Data is defined normally in a linear language from which someone can draw a two-dimensional description of the data. Two-dimensional layouts should be produced automatically just as two-dimensional flowcharts."

Sutherland, William R., Forgie, James W., and Morello, Marie V. "Graphics in Time-Sharing: a Summary of the TX-2 Experience." AFIPS, October 1969. (Quoted in Computing Reviews, November 1969.

"Ten years of experience in interactive computer graphics, with five of those years in a time-sharing environment, provides a unique source of material for this thorough and interesting look at lincoln Lab's TX-2 computer."

Walker, A.W. "An Interactive Graphical Debugging System." Computer Abstracts, February 1972.

"A system is described which provides an interactive graphical debugging facility for user programs. This system is implemented on an Adage AGT-10 and is operational for online debugging of higher-level language programs executing on an XDS 9300 host computer. System architecture and implementation are discussed. A formal definition of the DEBUG Command Language is given and a description of the utilization of the commands for program debugging is presented."

Weinwurm, George F. (Ed.) *On the Management of Computer Programming*. Auerbach, Princeton, N.J., 1970. (Quoted in *Computing Reviews*, May 1971.)

> "It comes as a distinct shock to the uninitiated that, for an activity that accounts for the expenditure of several billion dollars a year in the United States alone, the *management* of computer programming is still something of a black art."

Wirth, Niklaus. "Program Development by Stepwise Refinement." *Communications of the ACM*, April 1971.

> " . . . the student obtains the impression that programming consists mainly of mastering a language (with all the peculiarities and intricacies so abundant in modern PL's and relying on one's intuition to somehow transform ideas into finished programs. Clearly, programming courses should teach methods of design and construction, and the selected examples should be such that a gradual *development* can be nicely demonstrated."
>
> "This paper deals with a single example chosen with those two purposes in mind."
>
> "In each step, one or several instructions of the given program are decomposed into more detailed instructions. This successive decomposition or refinement of specifications terminates when all instructions are expressed in terms of an underlying computer or programming language . . ."
>
> "As tasks are refined, so the data may have to be refined, decomposed, or structured, and it is natural to *refine program and data specifications in parallel*."
>
> "A guideline in the process of stepwise refinement should be the principle to decompose decisions as much as possible, to untangle aspects which are only seemingly interdependent, and to defer those decisions which concern details of representation as long as possible. This will result in programs which are easier to adapt to different environments (languages and computers), where different representations may be required."
>
> "In the practical world of computing, it is rather uncommon that a program, once it performs correctly and satisfactorily, remains unchanged forever.

Usually its users discover sooner or later that their program does not deliver all the desired results, or worse, that the results requested were not the ones really needed. Then either an extension or a change of the program is called for, and it is in this case where the method of stepwise program design and systematic structuring is most valuable and advantageous. If the structure and the program components were well chosen, then often many of the constituent instructions can be adopted unchanged. Thereby the effort of redesign and reverification may be drastically reduced. As a matter of face, the adaptability of a program to changes in its objectives (often called maintainability) and to changes in its environment (nowadays called portability) can be measured primarily in terms of the degree to which it is neatly structured."

"The detailed elaborations on the development of even a short program form a long story, indicating that careful programming is not a trivial subject. If this paper has helped to dispel the widespread belief that programming is easy as long as the programming language is powerful enough and the available computer is fast enough, then it has achieved one of its purposes."

DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE

DCASM Final Report

APPENDIX B

CONCEPTUAL GROUPINGS PROGRAM FOR FORTRAN (GP-F)

(Prepared under Contract F19628-74-C-0061 by AMS, Inc.,
401 N. Harvard Avenue, Claremont, California 91711.)

DEVELOPMENTS IN COMPUTER AIDED SOFTWARE MAINTENANCE

DCASM Final Report

APPENDIX B

Table of Contents

APPENDIX B

CONCEPTUAL GROUPING PROGRAM FOR FORTRAN (GP-F)

1. GENERAL SYSTEM DESCRIPTION

As a very general rule, it is helpful to know what a person looks at, or perceives, when he works. Previous research has shown that a maintenance programmer tends to work with "conceptual groupings" in a programming language. Thus, the following suggestion has been made: The programmer could be helped in his work if he could be helped to recognize these "groupings." The Groupings Program for FORTRAN (GP-F) is intended to provide a way of experimentally pointing out the groupings to maintenance programmer. Thus it permits experimentation regarding the efficiencies of maintenance programming that might be gained if such a system were implemented operationally.

Functions of the GP-F may be specified as follows: Given a FORTRAN program as input, the GP-F should output a listing of that program, using various techniques (described later in Paragraph 2.2) to identify conceptual groupings of statements.

For experimentation, the GP-F should selectively apply (according to the researcher's specifications) various subsets of those techniques, to produce different patterns of groupings.

The GP-F should accept input either on cards, disc, or tape; and it should provide output on these or a line printer. It should also accept standard FORTRAN IV decks of any length and complexity, and provide output that is also standard FORTRAN. If compiled, the output should give results compatible with the compilation of the input deck.

2. FUNCTIONAL SPECIFICATIONS (GP-F)

2.1 Input/Output Specifications

The GP-F should allow input from a variety of sources and be able to provide output in the form of cards and listings. The program should allow for the user to determine the Input/Output specifications and the specifications which control the editing processes of the GP-F. The GP-F should allow multiple routines (i.e., Programs, Subroutines, and Functions) within the same input file. The program should process each routine within the input file independently of the other routines, resetting itself when initialized and after each routine. The GP-F might be designed to accept input files where END statements indicate the end of routines.

2.2 Program Functions

The GP-F should implement the following functions for identification of conceptual groups:

(1) It should identify and point out groupings characterized by like statement types.

(2) It should perform the following additional functions:

Print formats under each referencing I/O statement;

Sort declaratives to beginning of the program;

Indent nested do loops;

Mark transfer statements for easy identification;

Mark I/O statements.

2.3 Check for Like-Statement Groups

2.3.1 Description

This function should cause the program to look for statement groups that contain a certain percentage of like statements. A like-statement group would be a group of statements which contains a certain percentage of statements of one type. Given that a maximum size and minimum size of the like-statement group is known, the GP-F should determine whether a group of statements is a like-statement group by determining if any one statement type occurs within that group more than a specified percentage of the time. If a group had twenty statements, of which fifteen were "assignment" statements, and the criteria was 60% this group would be classified as a "Like-Statement group." Like-statement groups should include: I/O Transfer, and Assignment.

2.3.3 Implementation

The program should be preset with values for the maximum and minimum group size and the percentage of statements of one type that determines a Like-Statement group. The program could then scan the input in blocks of statements with the maximum size and determine the type of each statement. This would enable the calculation of type percentages which in turn would determine the existence of a "Like-Statement group."

2.4 Print Formats Under Each I/O Statement

2.4.1 Description

This function should cause format statements to be printed after EVERY input/output statement which references that format statement. Formats should not appear in the location they were in the original input deck unless that location followed a referencing I/O statement. All occurrences of the format statement except the first occurrence would appear as comments in the output listing (i.e., with a "C" in column 1).

2.4.2 Implementation

The formats in the program would be grouped into a table; whenever an I/O statement was processed the Format statement would be appended to the output file.

2.5 Sort Declaratives to the Beginning of the Program

2.5.1 Description

This function should cause all declaratives to be listed prior to the beginning of the edited listing, offset from the listing by blank lines. Declaratives should include: integer, real, common, dimension, double precision, complex, implicit, and data statements.

2.5.2 Implementation

Declaratives located during the first pass of the program, should be sorted and printed at top of listing.

2.6 Indent Nested Do Loops

2.6.1 Description

This function should cause nested do loops to be indented for each nested loop.

A simple DO loop begins with a DO statement and is terminated by a statement with the label (statement number) specified by the original DO statement. A nested DO loop contains more than one individual DO loop. This function should cause indentation to occur for EACH level in a NESTED DO loop.

2.6.2 Implementation

Start and end points of DO loops should be scanned by the program which would appropriately set an indent-controlling variable.

2.7 Mark Transfers

2.7.1 Description

This function should cause all conditional and unconditional transfer statements (If's, Goto's, Calls, etc.) to be marked. One method of marking would be to print a dotted line beneath the statement containing the transfer. Other methods might include: inserting blank lines before and/or after the transfer statement; overprinting the transfer statement; or printing a marker in the margin before the transfer statement.

2.7.2 Implementation

The program should search for transfer statements by examining input lines for keywords such as "IF," "GO TO," or "CALL" and "flag" these lines as containing transfer statements. This flagging could be done by using a table holding data on the lines in the input. The output routine for the FPP would then examine this table to determine the proper form and editing for the line containing the transfer statement when it is output.

2.8 Mark I/O Statements

2.8.1 Description

This function should cause all I/O operations to be marked. One method of marking would be to offset the I/O statement with blank lines before and after the statement. Other methods might include overprinting, indentation, or markers in the margin of the output.

2.8.2 Implementation

Similar to Transfers.

3. PROGRAM IMPLEMENTATION

3.1 Program Description

The prototype of the GP-F was designed, coded, tested, and implemented during the last four months of 1973. The language selected for this prototype was SITBOL, a modified version of SNOBOL, a language designed for string and character processing and manipulation. The SITBOL GP-F was run on the DECsystem-10 at the University of California at Irvine.

3.2 Operational Features

The SITBOL GP-F was designed to be easy to revise, and control. The program can be run in batch or timesharing mode on the DECsystem-10. Control of the program's operation, including the determination of input and output files, the control of the program's editing functions, and the control of various debugging features is available to the user via three methods:

(1) Default parameters;

(2) Parameters accepted from terminal;

(3) Parameters set from control disc file.

The input file for the program is specified by the user interactively. The output file is specified when the SITBOL system is initiated by the user. In the SITBOL/DEC system-10 environment, all I/O operations occur between the program and disc. Use of cards, line-printers, tape, etc. is enabled by use of a file transfer program.

The prototype GP-F allows the user to interactively specify certain control parameters that control the "mark Like-Statement groups" function. The user may specify, within certain limitations, the values of:

MAXSIZE - maximum group size

MINSIZE - minimum group size

and

PERCENT - the like-group determining percentage.

3.3 Prototype Differences from Specifications

3.3.1 Additional Features

In the prototype version of the GP-F, five additional functions or processes were added. These processes, controlled by the user in a manner similar to the other functions, enable various debugging routines and output control routines. These processes are:

(1) No Indent Statement Numbers. This switch causes the indenting of nested do loop statements to ignore statement labels, which remain near the left margin rather than "following" the statement when indented to the right.

(2) No Break Before Transfers. This switch inhibits the program from inserting a blank line before each transfer statement.

(3) Listing Identification. This switch causes the listing to contain information concerning program version, setting of control switches, and selection of editing parameters.

(4) Debugging Dump I/O. This switch causes the program to print on the output file the contents of the main buffer of the program

whenever an I/O operation occurs. This switch was a debugging aid during program development and testing.

(5) Debugging Dump Like-Statement Groups. This switch caused the program to print on the output file certain variables which were used in the search for Like-Statement Groups by one of the program's functions. This switch was a debugging aid during program development and testing.

3.3.2 Differences from Specifications

The prototype GP-F differs from the specifications in the following ways:

(1) The prototype GP-F does not sort declaratives by time when the "Sort Declaratives to the Beginning of the Program" function is specified. All declaratives are written at the top of the listing but they are not sorted by type.

3.3.3 Markings for Output Listing

(1) Transfer statements are marked by a "Dotted Line" printed beneath the transfer statement in the output listing.

(2) I/O statements are marked by a preceding blank line in the output listing.

(3) Like-Statement groups are marked by a "Dashed Line" before and after the statements forming the group.

(4) Output Form. The program does not check for splitting groupings across page boundaries in the case of lineprinter output.

3.4 Operational Deficiencies and Bugs

(1) The program does not in all cases prepare output that is directly accepted by a FORTRAN compiler. This is because some editing functions that insert blank lines or various markings in the output file do not precede such lines with "C's" to indicate comment statements.

APPENDIX B

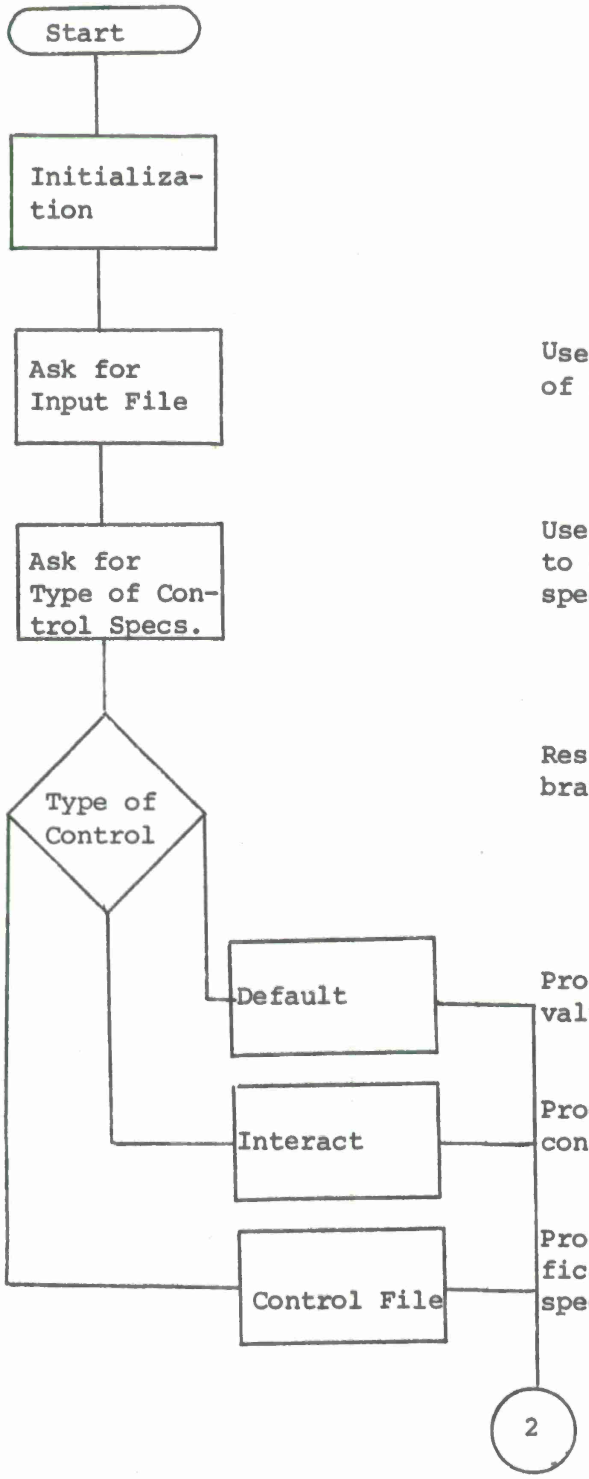
CONCEPTUAL GROUPINGS PROGRAM FOR FORTRAN (GP-F)

4. CONCEPTUAL GROUPING PROGRAM FOR FORTRAN (GP-F)

FLOW CHARTS

CONCEPTUAL GROUPING PROGRAM FOR FORTRAN (GP-F)

FLOW CHARTS



User asked to supply name of input file.

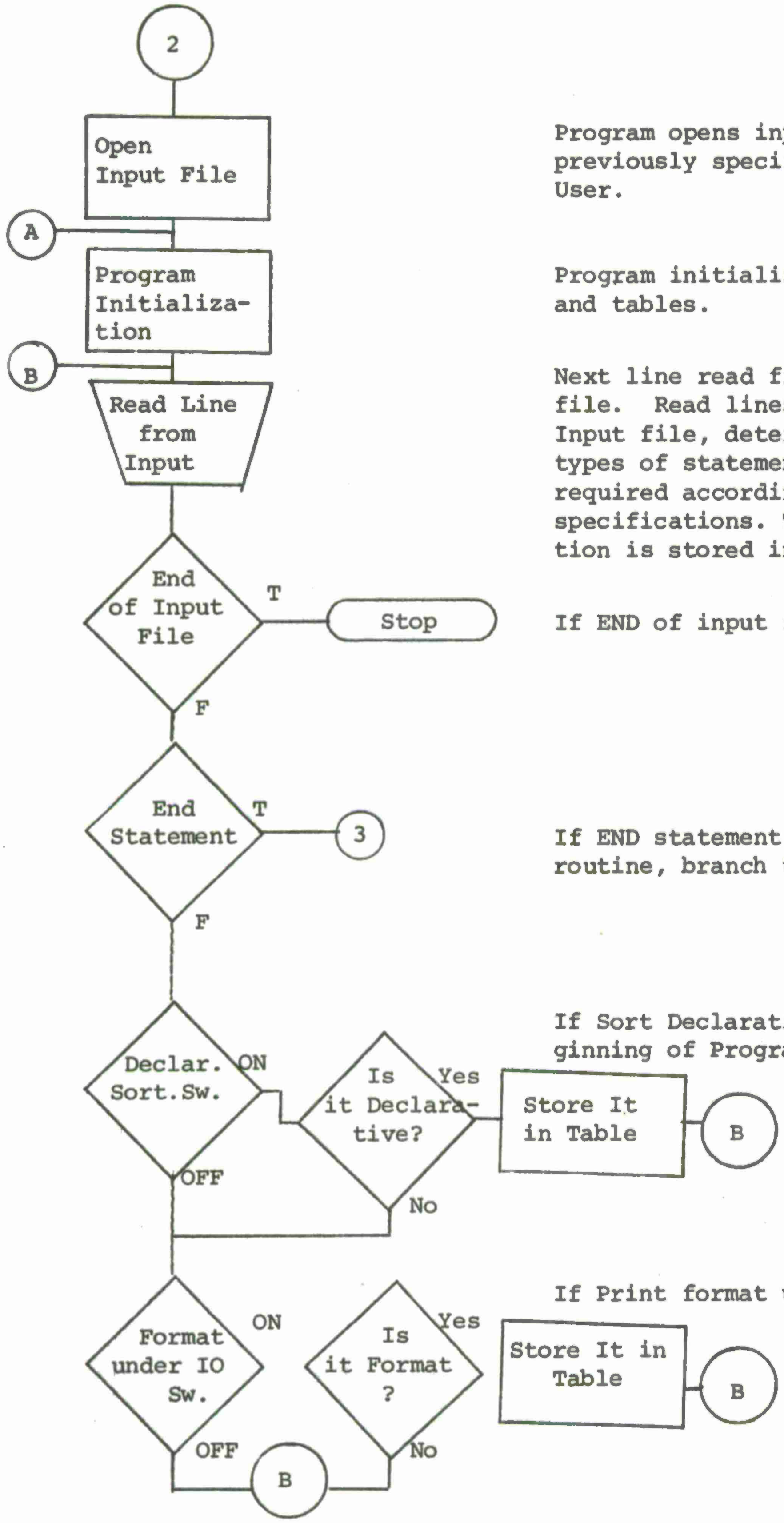
User asked how program is to set control specifications.

Result of above determines branch

Program assumes default values.

Program asks User to specify control specifications.

Program reads control specifications from control file specified by User.



Program opens input file previously specified by User.

Program initializes variables and tables.

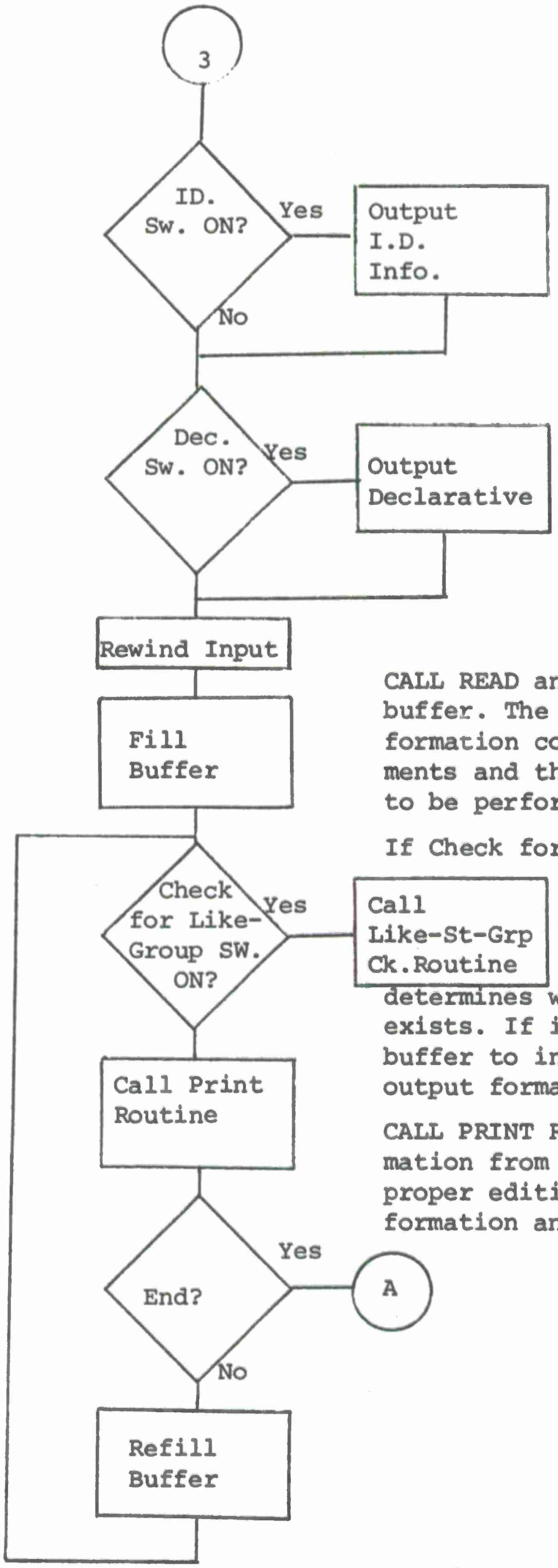
Next line read from Input file. Read lines from the Input file, determine what types of statement and editing required according to control specifications. This information is stored in the BUFFER.

If END of input file, STOP.

If END statement, terminating routine, branch to 3.

If Sort Declaratives to Beginning of Program Switch is on; if statement read is declarative, store it in table.

If Print format under every I/O Switch is on; if statement is format store it in table.



If Print I.D. Switch is on, OUTPUT heading and Program information.

If Sort Declarative Switch is on, output declaratives from table

Rewind input file.

CALL READ and EDIT routines to fill buffer. The BUFFER contains all the information concerning the type of statements and the editing functions that are to be performed during OUTPUT.

If Check for Like-Statement switch is on, call routine to CHECK FOR LIKE-STATEMENTS GROUP. This routine examines the statements in the buffer and determines whether a LIKE-STATEMENT GROUP exists. If it does it adds or changes the buffer to include the instructions for output formatting and editing.

CALL PRINT ROUTINE. Outputs the information from the buffer performing the proper editing as specified by the information and statements in the buffer.

If End, branch to A.

REFILL buffer by calling READ and EDIT routines.

APPENDIX B

CONCEPTUAL GROUPINGS PROGRAM FOR FORTRAN (GP-F)

5. PROGRAM LISTING OF GP-F IN SITBOL

```
* THE FOLLOWING ARE SOME BROAD DEFINITIONS TO ASSIST CONVERSION
* OF THIS PROGRAM FROM SITBOL TO AN ASSEMBLY LEVEL LANGUAGE
*
* INTRODUCTION --
* THIS PROGRAM WAS WRITTEN IN UCI SITBOL ON THE DECSYSTEM 10 AT UCI
* A NUMBER OF ADDITIONS TO THIS PROGRAM DURING DEVELOPMENT
* AND TESTING ARE SPECIFIC TO THOSE OPERATIONS AND
* NEED NOT BE INCLUDED IN ANY CONVERSION.  THERE IS A QUESTION AT
* THIS TIME HOW THE EXACTLY CONVERSIN WILL BE DONE AND IF THE
* EDITING CONTROL SWITCHES (IDENTIFIED BY THE SUFFIX .SW) WILL BE
* IMPLEMENTED
*
* THE PARTS OF THE PROGRAM WHICH PROBABLY WILL NOT BE INCLUDED IN ANY
* CONVERSION ATTEMPT WILL BE THE FOLLOWING:
*
* ROUTINES TO ALLOW SELECTION OF SWITCHES TO CONTROL EDITINXG VIA
* THREE DISTINCT METHODS. (STATEMENTS 230-293) (STATEMENTS 9195-9498)
* SOME OTHER METHOD OF SWITCH SPECIFICATION WILL HAVE TO BE ADDED
* TO REPLACE THESE SECTIONS
*
* SOME OF THE SECTIONS THAT ALLOWED SELECTIVE DUMPING DURING TESTING
* HAVE BEEN REMOVED.  THESE SECTIONS IDENTIFIED WITH THE SWITCHES:
* P2DUMP.SW AND CGDUMP.SW ARE UNNECESSARY
*
* ALL SECTIONS PROVIDING PROGRAM IDENTIFICATION MAY BE IDENTIFIED BY THE SWITCH:
* ID.SW
* THESE ARE PRIMARILY LOCATED IN STATEMENTS 700-795 AND 2915-2955.
**
* END OF DOCUMENTATION FOR CONVERSION
*
*-------------------------------------------------------------------------------------
*
*
* THIS IS NEW FORTRAN POSTPROCESSOR, OVERTON PROJECT
*
* THE FOLLOWING DELETES TRAILING SPACES FROM ALL INPUT DATA
        &TRIM = 1
* SET INPUT VARIABLES TO BEGIN PROGRAM
*
* THE FOLLOWING DESIGNATES A PATTERN OF DIGITS
        NUMBERS = '0123456789'
* THE FOLLOWING THRU 165 DEFINITIE PATTERNS FOR STATEMENT IDENTIFICATION
        DEC.P = 'INTEGER' ! 'DIMENSION' ! 'REAL' ! 'COMMON' ! 'DATA'
        TRN.P = 'IF' ! 'GOTO' ! 'GO TO' ! 'CALL'
        I.P = 'READ' ! 'INPUT' ! 'ACCEPT' ! 'REREAD'
        O.P = 'PRINT' ! 'WRITE' ! 'TYPE'
        IO.P = I.P ! O.P
        NI.P = 'CONTINUE' ! 'STOP' ! 'END' ! 'PROGRAM'
* MARGIN IS THE INDENT DISTANCE FOR DO LOOP INDENTATION
        MARGIN = '      '
*
**********************************************************
* LAST UPDATE 10-28-73 SUNDAY  11:00 AM
* ALL FIXES AS PER OPLOG.INF IN PLACE
*
        EDITION = 'EDITION 4.0   11-20-73'
*
* PRESET VARIABLES
*
* DEFINITIONS OF SWITCH CONDITIONS (OFF/ON)
```

```
Ø13ØØ          OFF = Ø
Ø132Ø          ON = 1
Ø134Ø   * TOP IS THE TOP OF THE DO LOOP STACK
Ø136Ø          TOP = Ø
Ø138Ø   * FRM,COUNT IS THE FORMAT COUNT (PASS ONE)
Ø14ØØ          FRM,COUNT = Ø
Ø142Ø   * LINEC IS THE NUMBER OF INPUT LINES (PASS ONE)
Ø144Ø   * A FIX WILL BE INSERTED SO CONTINUATION LINES WILL NOT BE COUNTED
Ø146Ø          LINEC = Ø
Ø148Ø   * DEC,COUNT COUNTS THE DECLARITIVES (PASS ONE)
Ø15ØØ          DEC,COUNT = Ø
Ø152Ø   * DEFINITION FOR \SWCOND,FROM (SELECTION)
Ø154Ø          SWCOND,FROM = 'DEFAULTED'
Ø156Ø   *
Ø158Ø   * TABLES AND ARRAYS *********
Ø16ØØ   *
Ø162Ø   * HOLDS FORMATS, REFERENCE IS FORMAT = FORMATS<F. NO.>
Ø164Ø          FORMATS = TABLE()
Ø166Ø   * HOLDS DECLARATIVES
Ø168Ø          DECLARS = ARRAY(2Ø)
Ø17ØØ   * HOLDS STACK FOR DO LOOP NESTS
Ø172Ø          STACK = ARRAY(1Ø)
Ø174Ø   * ***
Ø176Ø   * THE FOLLOWING TWO STATEMENTS CONTROL BUFFER SIZE
Ø178Ø   * BUFFER IS CURRENTLY 2Ø
Ø18ØØ   * BUFFER IS 2Ø X 5 (5 IS CONSTANT WIDITH OF BUFFER)
Ø182Ø          B,SIZE = 2Ø
Ø184Ø          B,T = ARRAY('2Ø,5')       :(INSW,RTJ)
Ø186Ø   * THE FOLLOWING THRU 295 IS STRICTLY PDP-10 SITBOL AND IS ONLY INCLUDED FOR
Ø188Ø   * EASE OF USE DURING TESTS, IT SETS SWITCH CONDITIONS VIA ONE OF THREE METHODS
Ø19ØØ   READTTY,Ø      TTY = 'SWITCH SELECTION (TYPE IN- FILE,DEFAULT OR TTY)'
Ø192Ø   READTTY,1      NLINE = TTY
Ø194Ø          IDENT(NLINE,'FILE')       :S(SWCARD,FILE)
Ø196Ø           IDENT(NLINE,'DEFAULT') :S(D,F)
Ø198Ø          IDENT(NLINE,'TTY')        :S(SWCARD,TTY)
Ø2ØØØ          TTY = 'LEGAL REPLIES ARE- FILE,DEFAULT, OR TTY' :(READTTY,1)
Ø2Ø2Ø   * ROUTINE TO ASK FOR INPUT FILE NAME AND THEN SET SWITCH DEFAULTS
Ø2Ø4Ø   INSW,RTJ                 TTY = 'ENTER INPUT FILE NAME'
Ø2Ø6Ø          NLINE = TTY
Ø2Ø8Ø          IDENT(NLINE,'')           :S(FILE,DEF)
Ø21ØØ          FILE.NAME = NLINE         :(SETSW,DEF)
Ø212Ø   FILE.DEF       FILE.NAME = 'FTN.SIT'
Ø214Ø          TTY = 'FILE NAME DEFAULTED'
Ø216Ø   SETSW,DEF      TTY =
Ø218Ø          INPUT('INF',FILE.NAME)
Ø22ØØ          TTY = 'INPUT FILE: ' FILE,NAME '  OPENED.'
Ø222Ø          FORM,SW = ON
Ø224Ø          DEC,SW = ON
Ø226Ø          DO,SW = ON
Ø228Ø          TRN,SW = ON
Ø23ØØ          IO,SW = ON
Ø232Ø          CG.SW = ON
Ø234Ø          INSNO,SW = ON
Ø236Ø          P2DUMP,SW = OFF
Ø238Ø          SPBFGOTO.SW = OFF
Ø24ØØ          CGDUMP,SW = OFF
Ø242Ø   *
Ø244Ø          ID.SW = ON
Ø246Ø          CG.MAX = 1Ø
Ø248Ø          CG,SIZE = 7        :(READTTY,Ø)
```

```
*
* ************************************************************************
* THE FOLLOWING ARE THE DEFINITIONS OF THE VARIOUS FUNCTIONS
*  UTILIZED IN THIS PROGRAM
*
* PRINT,RTJ      ATTEMPTS TO PRINT BUFFER (1 THRU B,MARK)
* (USES PRINT,ST FOR OUTPUT)
*
* PRINT,ST       ATTEMPTS TO PRINT ONE STATEMENT (L,NO) USING
*                THE VARIOUS SWITCHES AND INFORMATION IN THE
*                BUFFER TO PROPERLY FORMAT THE OUTPUT
*
* REDIT,L        READS ONE INPUT LINE AND EDITS IT PLACING THE
*                INFORMATION IN BUFFER LOCATION (B,LOC), CAN FAIL,
*
* FILL.B         FILLS BUFFER,  FIRST MOVES REMAINDER OF BUFFER TO TOP
*                OF BUFFER, AND FINALLY FILLS BUFFER RETURNING
*                BUFFER SIZE AS B,SIZE,  B,SIZE WILL BE 10 EXCEPT
*                WHEN REDIT,L READS EOF FROM INPUT FILE
*        (THIS ROUTINE USES REDIT,L FOR INPUT)
*
* CHECK4.CG      THIS ROUTINE CHECKS FOR THE EXISTENCE OF A
*                CONCEPTUAL GROUP IN THE BUFFER OF MINIMUM SIZE
*                CG,MIN AND RETURNS START POINT (ST,PT) AND
*                END POINT (END,PT) OF CG IF IT EXISTS,  THIS
*                ROUTINE FAILS IF NO CONCEPTUAL GROUP EXISTS,
*
* STMNT,NO       THIS ROUTINE ELIMINATES BLANKS FROM A STRING
*                WHICH IS USUALLY THE FIRST 6 COLUMNS OF AN
*                INPUT FIELD
*
* BLANK          OUTPUTS (NOOF,L) BLANK LINES
*
*
**********************************************************************************
* DEFINE FUNCTIONS ********************
*
D,F     DEFINE('REDIT,L(B,LOC)')
*
        DEFINE('PRINT,RTJ(B,MARK)')
*
        DEFINE('FILL.B(B.MARK2)')
*
        DEFINE('PRINT,ST(L,NO)')
*
        DEFINE('CHECK4,CG(ST,PT)')
*
        DEFINE('SYMNT,NO(STRING)')
*
        DEFINE('BLANK(NOOF,L)')
*
*
*
******* END OF FUNCTION DEFINITIONS *********
***************************************************************************
*
* THE FOLLOW OUTPUTS THE SWITCH CONDITIONS IS SPECIFIC FOR PDP-10
* AND FPP TEST CONDITIONS
* SWITCH CONDITION OUTPUT
        EQ(IO,SW,OFF)                       :S(READ1)
        OUTPUT = DUPL(' ',30) DUPL('*',49)
        OUTPUT = DUPL(' ',30) '****    S W I T C H   C O N D I T I O N S    ****'
        OUTPUT = DUPL(' ',30) DUPL('*',49)
        OUTPUT = 'SWITCH CONDITIONS ENTERED FROM ' SWCOND,FROM
        OUTPUT =
        OUTPUT = 'CONCEPTUAL GROUP PARAMETERS ARE -'
        OUTPUT =
        OUTPUT = 'MAX = ' CG,MAX '   GROUP SIZE FACTOR IS ' CG,SIZE
        OUTPUT =
        OUTPUT = ' THE FOLLOWING SWITCHES ARE OFF -'
        OUTPUT =
  OUTPUT = EQ(FORM,SW,OFF) 'FORMAT PRINT WITH IO'
  OUTPUT = EQ(CG.SW,OFF) 'CONCEPTUAL GROUPS'
  OUTPUT = EQ(DEC,SW,OFF)  'DECLARATIVE SORT TO BEGINNING'
  OUTPUT = EQ(INSNO,SW,OFF) 'INDENT STATEMENT NUMBERS'
  OUTPUT = EQ(DO,SW,OFF) 'DO GROUPING'
  OUTPUT = EQ(SPBFGOTO,SW,OFF) 'SPACE BEFORE TRANSFERS'
  OUTPUT = EQ(TRN,SW,OFF) 'TRANSFER GROUPINGS'
  OUTPUT = EQ(IO.SW,OFF) 'IO GROUPING'
        MISC = BLANK(3)
*
*****************************************************************************************
* DESCRIPTION OF M,T ARRAY:
*       SIZE IS CG,MAX BY 5 (CG,MAX,5) 5 PARAMETERS ARE=-
*
*       1- STATEMENT NUMBER (6 COLUMNS)
*       2- LINE WITHOUT STATEMENT NUMBER (7-72, TRIM)
*       3- TYPE (IO,TR,MI,DE,DO,AS,FO)
*       4-      1- SKIP BLANK LINE BEFORE
*               2- SKIP LINE AFTER
*               3- SKIP LINE BEFORE AND AFTER
*               10 - BEGIN CG GROUP MARKER
*               20 - END CG GROUP MARKER
*       5- INDENTATION DISTANCE OR DEGREE
*****************************************************************************
*
****************************
*** PASS ONE OPERATIONS  ***
****************************
*
* THIS ROUTINE READS THE ENTIRE INPUT FILE IDENTIFYING AND
* STORING ALL FORMATS AND DECLARATIVES,  FORMATS ARE PLACED IN A TABLE
* NAMED 'FORMATS' ACCORDING TO THEIR FORMAT NUMBERS (REFERENCED BY),
*****
* THE NEXT STATEMENT TRANSFER ONE LINE OF DATA <80 COL> FROM THE
* INPUT FILE <INF> TRIMS EXCESS RT BLANKS AND PUTS IT IN LINE
* AN END OF FILE CONDITIONS CAUSES A TRANSFER TO 'PASSTWO'
READ1   LINE = INF                              :F(PASSTWO)
*
* INCREMENT LINE COUNT BY ONE
        LINEC = LINEC + 1
* COPY LINE TO NLINE
        NLINE = LINE
*
* EXTRACT COLUMNS 1-6
        LINE LEN(6) . LABEL
        NLINE LABEL =
        INDEX = STMNT,NO(LABEL)
* AT THIS POINT LABEL CONTAINS THE STATEMENT NUMBER IF ANY DEVOID OF BLANKS
* WHILE NLINE CONTAINS COLUMNS 7-80 (TRIMED RIGHT) OF THE INPUT
```

```
Ø49ØØ   * FORMAT - CHECK IF THIS IS A FORMAT, BY PATTERN MATCHING
Ø492Ø   *
Ø494Ø   * IF IT'S NOT A FORMAT TRANSFER TO DEC,SEARCH
Ø496Ø           NLINE 'FORMAT'                        !F(DEC,SEARCH)
Ø498Ø   * INCREMENT FORMAT COUNT BY ONE
Ø5ØØØ           FRM,COUNT = FRM,COUNT + 1
Ø5Ø2Ø   * STORE COL 7-8Ø (TRIMMED) WHICH IS THE FORMAT IN THE TABLE
Ø5Ø4Ø   * 'FORMATS' REFERENCED BY THE FORMAT NUMBER (INDEX)
Ø5Ø6Ø   * THEN TRANSFER TO THE READ STATEMENT FOR MORE INPUT
Ø5Ø8Ø           FORMATS<INDEX> = LINE    !(READ1)
Ø51ØØ   *
Ø512Ø   * PATTERN MATCH FOR DECLARATIVE, NON DECLARATIVE GO TO READ1 FOR MORE INPUT
Ø514Ø   DEC,SEARCH      NLINE DEC,P                   !F(READ1)
Ø516Ø   * INCREMENT DEC,COUNT BY ONE
Ø518Ø           DEC,COUNT = DEC,COUNT + 1
Ø52ØØ   * SAVE DECLARATIVE IN ARRAY DECLARS<DEC,COUNT>
Ø522Ø           DECLARS<DEC,COUNT> = LINE             !(READ1)
Ø524Ø   *
Ø526Ø   *
Ø528Ø   ***************************************
Ø53ØØ   *** PASS TWO IDENTIFICATION         ***
Ø532Ø   ****************************************
Ø534Ø   * THIS IDENTIFICATION IS NOT REQUIRED AND IS DEPENDENT ON
Ø536Ø   * CONDITION OF ID,SW (IDENTIFICATION SWITCH)
Ø538Ø   *
Ø54ØØ   PASSTWO         MISC = EQ(ID,SW,OFF)     !S(P,2)
Ø542Ø           MISC = BLANK(5)
Ø544Ø   PASS.2  O,LINE = 'FPP ' EDITION DUPL(' ',12)
Ø546Ø     OUTPUT = O.LINE 'EDITED LISTING OF FILE ' FILE,NAME ' RUN ON ' DATE()
Ø548Ø           OUTPUT =
Ø55ØØ           OUTPUT = DUPL('*',12Ø)
Ø552Ø           OUTPUT =
Ø554Ø   *
Ø556Ø   *
Ø558Ø   *
Ø56ØØ   ********************************
Ø562Ø   *** PASS TWO OPERATIONS BEGIN ***
Ø564Ø   ********************************
Ø566Ø   * THE FOLLOWING STATEMENTS (THRU 294Ø) ARE SITBOL DEPENDENT THEY FUNCTION
Ø568Ø   * TO REWIND THE INPUT FILE SO IT CAN BE REREAD BY PASS TWO
Ø57ØØ   P,2     REWIND(FILE,NAME)
Ø572Ø           DETACH('INF')
Ø574Ø           INPUT('INF',FILE,NAME)                !F(ERROR)
Ø576Ø   *
Ø578Ø   *
Ø58ØØ   * IF THE DEC.SW (DECLARATIVE PRINT BEGINNING)
Ø582Ø   * IS OFF THE DECLARATIVES ARE NOT PRINTED BY THEMSELVES AT THE
Ø584Ø   * BEGINNING OF THE OUTPUT LISTING
Ø586Ø   *
Ø588Ø   * THE FOLLOWING ROUTINE LOOPS TO PRINT ALL DECLARATIVES (FROM 1 TO
Ø59ØØ   * DEC,COUNT FROM THE ARRAY DECLARS
Ø592Ø   *
Ø594Ø           MISC = EQ(DEC,SW,OFF)                 !S(FB,ST)
Ø596Ø   *
Ø598Ø           I = Ø
Ø6ØØØ   DECOUNT,AD1     I = I + 1
Ø6Ø2Ø           OUTPUT = LE(I,DEC,COUNT) DECLARS<I> !S(DECOUNT,AD1)
Ø6Ø4Ø   *
Ø6Ø6Ø   *
Ø6Ø8Ø   *PRINT A BLANK LINE
```

```
DECOUT,FIN      OUTPUT =
*
*
*
* THE BUFFER MUST BE FILLED PRIOR TO ENTERING THE MAIN
* OPERATIONAL SECTION OF PASS TWO\
* STATEMENTS 4015 - 4040 DO THIS.
* FILL BUFFER
FB,ST    I = 0
FB,AD1   I = I + 1
          REDIT,L(I)                          :F(ERROR)
          MISC = EQ(I,CG,MAX)      :F(FB,AD1)
*
* BEGIN MAIN OPERATIONAL SECTION OF PASS TWO
* ASSUME HERE BUFFER IS FULL SO BEGIN CYCLE
* CHECK BUFFER FOR CG IF SWITCH IS ON
* IF CG SWITCH IS ON, THE BUFFER WILL BE CHECKED FOR
* A CONCEPTUAL GROUP BY CHECK4,CG ROUTINE, IF CG SWITCH IS OFF
* TRANSFER TO NO,CG
CHK,CG   MISC = EQ(CG.SW,OFF)                 :S(NO,CG)
*
* NEXT STATEMENT CALLS CHECK4,CG (IGNORE MISC)
* IF NO CONCEPTUAL GROUP EXISTS,THE RETURN WILL CAUSE
* A TRANSFER TO NO,C
          MISC = CHECK4,CG(MISC)              :F(NO,CG)
* AT THIS POINT WE KNOW THAT CHECK4,CG FOUND A CONCEPTUAL GROUP AND MARKED
* ITS BEGINNING POINT AND ENDING POINT IN BUFFER (ST.PT, END,PT)
*
* NOW SAVE IN BUFFER PARAMETERS TO MARK CG WHEN OUTPUT OCCURS\
* MARK CG BEGINNING AND ENDING
*
          M,T<ST,PT,4> = 10
          M,T<END,PT,4> = 20
*
* NOW CALL PRINT.RTJ TO PRINT THE PART OF THE BUFFER CONTAINING
* THE CONCEPTUAL GROUP, IN THIS CASE 1-END,PT ARE PRINTED
* THE REMAIN STATEMENTS IN THE BUFFER WILL BE MOVED TO THE TOP
          MISC = PRINT.RTJ(END,PT)
*
* CALL FILL,B TO REFILL TO BUFFER FROM THE END,PT
          MISC = FILL,B(END,PT)
* AFTER CALLING FILL.B WE MUST CHECK TO SEE IS THE BUFFER COULD NOT BE
* FILLED BECAUSE AN END-OF-FILE WAS REACHED DURING INPUT
*IF THIS IS THE C\AE TRANSFER TO FIN.OUT, ELSE
* BEGIN CYCLE AGAIN BY TRANSFERING TO CHK,CG
          MISC = LT(B,SIZE,CG,MAX)            :F(CHK,CG)S(FIN,OUT)
*
*
* NO CHECK FOR CG OR NO CG SO PRINT ENTIRE BUFFER BY CALLING PRINT,RTJ
NO.CG    MISC = PRINT.RTJ(CG,MAX)
*
* IF LAST TIME BUFFER WAS FILLED AN END OF FILE WAS ENCOUNTERED DURING
* INPUT THE NEXT STATEMENT WILL CAUSE A TRANSFER TO FIN,OUT
* OTHERWISE IT WILL TRANSFER TO CALLFB
*
*
REFILL   MISC = LT(B,SIZE,CG,MAX) :F(CALLFB)
*
* ALL INPUT COMPLETED, NOW PERFORM PRINT,RTJ FOR REMAINDER OF
* BUFFER (1-B,SIZE) THEN TRANSFER TO FINIS
```

```
07300    * FINAL OUTPUT THEN END
07320    FIN,OUT          MISC = PRINT,RTJ(B,SIZE)          :(FINIS)
07340    *
07360    *
07380    * REFILL THE BUFFER BY CALLING FILL,B THEN RECYLE TO CHK.CG
07400    CALLFB           MISC = FILL,B(B,SIZE)    :(CHK,CG)
07420    *
07440    *
07460    ****************************************
07480    * ALL DONE SO PRINT A BLANK LINE AND TERMINATE PROGRAM
07500    * WITH TRANSFER TO END
07520    FINIS   OUTPUT =          :(END)
07540    ****************************************
07560    *
07580    *******************************************************
07600    ***** FUNCTIONS FOR PROGRAM                      ******
07620    *******************************************************
07640    *
07660    REDIT,L          LINE = INF                    :F(FRETURN)
07680            NLINE = LINE
07700            &ANCHOR = 1
07720            NLINE 'C'         :F(NO,COMMENT)
07740            M,T<B,LOC,3> = 'CO' ; &ANCHOR = 0
07760            M,T<B,LOC,2> = LINE                    :(RE,EN)
07780    NO,COMMENT       LINE LEN(6) . LABEL
07800            INDEX = STMNT,NO(LABEL)
07820            NLINE LABEL =
07840            M,T<B,LOC,1> = LABEL
07860            M,T<B,LOC,2> = NLINE
07880            M,T<B,LOC,4> = 0
07900    *
07920    ISIT.DO          NLINE 'DO'                    :F(ISIT,IO)
07940            M,T<B,LOC,3> = 'DO'
07960            MISC = EQ(DO.SW,OFF)                   :S(RE,EN)
07980            M,T<B,LOC,4> = 3
08000            INDENT = MARGIN INDENT
08020            NLINE BREAK(NUMBERS) SPAN(NUMBERS) . NO
08040            NO BREAK(NUMBERS) =
08060            TOP = TOP + 1
08080            STACK<TOP> = NO                        :(RE.EN)
08100    *
08120    ISIT.IO          NLINE IO.P                    :F(ISIT,TR)
08140            M,T<B,LOC,3> = 'IO'
08160            MISC = EQ(IO.SW,OFF)                   :S(RE,EN)
08180            M,T<B,LOC,4> = 1                       :(RE.EN)
08200    *
08220    ISIT.TR          NLINE TRN,P                   :F(ISIT,FO)
08240            M,T<B,LOC,3> = 'TR'
08260            MISC = EQ(TRN,SW,OFF)                  :S(RE,EN)
08280            M,T<B,LOC,4> = 3
08300            NLINE 'GO TO'                          :F(RE,EN)
08320            M,T<B,LOC,4> = 20                      :(RE.EN)
08340    *
08360    ISIT.FO          NLINE 'FORMAT'                :F(ISIT,DEC)
08380            M,T<B,LOC,3> = 'FO'                    :(RE.EN)
08400    *
08420    ITIS.AS          M,T<B,LOC,3> = 'AS'           :(RE.EN)
08440            M,T<B,LOC,3> = 'AS'                    (RE,EN)
08460    *
08480    ISIT.MI          NLINE MI.P                    :F(ITIS,AS)
```

```
08500            M,T<B,LOC,3> = 'MI'                 :(RE,EN)
08520   *
08540   ISIT.DEC          NLINE DEC,P                :F(ISIT,MI)
08560                     M,T<B,LOC,3> = 'DE'        :(RE,EN)
08580   *
08600   *
08620   RE.EN    M,T<B,LOC,5> = INDENT
08640            EQ(TOP,0)                           :S(RE,END)
08660            IDENT(LABEL,'        ')   :S(RE,END)
08680   *
08700            IDENT(STACK<TOP>,STMNT,NO(LABEL))        :F(RE,END)
08720   *
08740            INDENT MARGIN =
08760            TOP = TOP - 1
08780            LT(M,T<B,LOC,4>,1)                  :F(RE,CON6)
08800            M,T<B,LOC,4> = 2                    :(RE,END)
08820   RE.CON6           M,T<B,LOC,4> = M,T<B,LOC,4> + 1
08840   *
08860   RE.END            O,K = 6                    :(RETURN)
08880   *
08900   *
08920   ***************************************************************
08940   *
08960   *
08980   PRINT,RTJ         I = 0
09000   PR.AD1            I = I + 1
09020                     GT(I,B,MARK)               :S(RETURN)
09040                     PRINT,ST(I)                :(PR,AD1)
09060   *
09080   ***************************************************************
09100   *
09120   CHECK4,CG  CTO,PT = CG,SIZE + B,SIZE / 10
09140            TR,COUNT = 0; IO,COUNT = 0; AS,COUNT = 0
09160            I = 0
09180   C4,AD1   I = I + 1
09200            LT(I,B,SIZE)              :F(C4,END)
09220            IDENT(M,T<I,3>,'TR')      :F(C4,C1)
09240            TR,COUNT = TR,COUNT + 1 :(C4,AD1)
09260   C4,C1    IDENT(M,T<I,3>,'IO')      :F(C4,C2)
09280            IO,COUNT = IO,COUNT + 1 :(C4,AD1)
09300   C4,C2    IDENT(M,T<I,3>,'AS')      :F(C4,AD1)
09320            AS,COUNT = AS,COUNT + 1 :(C4,AD1)
09340   *
09360   *
09380   C4,END   TYP = GT(TR,COUNT,CTO,PT) 'TR'  :S(CG,FOUND)
09400            TYP = GT(IO,COUNT,CTO,PT) 'IO'   :S(CG,FOUND)
09420            TYP = GT(AS,COUNT,CTO,PT) 'AS'  :S(CG,FOUND)
09440   *
09460            ST,PT = 0                 :(CHK,FAIL)
09480   *
09500   CG.FOUND          I = 0
09520   CG4,AD1           I = I + 1
09540            IDENT(M,T<I,3>,TYP)       :F(CG4,AD1)
09560            ST,PT = I
09580   *
09600            I = B,SIZE + 1
09620   CG4,M1   I = I - 1
09640            IDENT(M,T<I,3>,TYP)       :F(CG4,M1)
09660            END,PT = I
09680            TTY = 'SUCCESS ' ST,PT ' ' END,PT :(RETURN)
```

```
Ø97ØØ   *
Ø972Ø   *
Ø974Ø   ** NEXT TWO STATEMENTS NEVER EXECUTE ??????
Ø976Ø   CHK,FAIL          O,K = 27                     :(FRETURN)
Ø978Ø   *
Ø98ØØ   *
Ø982Ø   *
Ø984Ø   FILL,B            LT(B,MARK2,1)   :S(ERROR)
Ø986Ø   *
Ø988Ø            M = B,MARK2
Ø99ØØ            I = Ø
Ø992Ø   FIL,AD1               M = M + 1
Ø994Ø            MISC = GT(M,CG,MAX)                   :S(FIL,CON2)
Ø996Ø            I = I + 1
Ø998Ø            M,T<I,1> = M.T<M,1>
1ØØØØ            M,T<I,2> = M.T<M,2>
1ØØ2Ø            M,T<I,3> = M.T<M,3>
1ØØ4Ø            M,T<I,4> = M.T<M,4>
1ØØ6Ø            M,T<I,5> = M.T<M,5>                   :(FIL,AD1)
1ØØ8Ø   *
1Ø1ØØ   FIL,CON2          I = I + 1
1Ø12Ø                     GT(I,1Ø)                     :S(FIL,CON4)
1Ø14Ø            MISC = REDIT,L(I)                     :S(FIL,CON2)
1Ø16Ø   * REDIT HAS FAILED
1Ø18Ø            B,SIZE = I - 1                        :(RETURN)
1Ø2ØØ   *
1Ø22Ø   FIL,CON4          B,SIZE = 1Ø                  :(RETURN)
1Ø24Ø   *
1Ø26Ø   ***********************************************************
1Ø28Ø   *
1Ø3ØØ   STMNT,NO          STRING ' ' =                 :S(STMNT,NO)
1Ø32Ø            STMNT,NO = STRING                     :(RETURN)
1Ø34Ø   *
1Ø36Ø   *
1Ø38Ø   ***********************************************************
1Ø4ØØ   *
1Ø42Ø   BLANK    I = Ø
1Ø44Ø   BL,AD1            I = I + 1
1Ø46Ø            OUTPUT =
1Ø48Ø            MISC = LT(I,NOOF.L)      :F(RETURN)S(BL,AD1)
1Ø5ØØ   *
1Ø52Ø   *
1Ø54Ø   ***********************************************************
1Ø56Ø   ************************************************************
1Ø58Ø   *
1Ø6ØØ   PRINT,ST          O,K = 4
1Ø62Ø            EQ(END,FLAG,ON) :S(RETURN)
1Ø64Ø             MISC = IDENT('END',STMNT,NO(M,T<L,NO,2>)) :F(PST,1)
1Ø66Ø            END,FLAG = ON
1Ø68Ø   PST,1    MISC = IDENT(M,T<L,NO,3>,'FO')            :F(PST,2)
1Ø7ØØ            MISC = EQ(FORM,SW,OFF)  :S(PST,11)F(RETURN)
1Ø72Ø   PST,2    MISC = IDENT(M,T<L,NO,3>,'CO')  :F(PST,3)
1Ø74Ø            OUTPUT = M,T<L,NO,2>            :(PST,25)
1Ø76Ø   *
1Ø78Ø   PST,3    MISC = IDENT(M,T<L,NO,3>,'DE')  :F(PST,5)
1Ø8ØØ            MISC = EQ(DEC,SW,OFF)   :S(PST,11)F(RETURN)
1Ø82Ø   *
1Ø84Ø   * CHECK FOR CG GROUP
1Ø86Ø   PST,5    OUTPUT = ED(M,T<L,NO,4>,1Ø) DUPL('-',1ØØ) :F(PST,7)
1Ø88Ø   *
```

```
10900           OUTPUT = ' CONCEPTUAL GROUP '
10920           OUTPUT =
10940   PST,7              MISC = EQ(M,T<L,NO,4>,1)           :S(PST,10)
10960   *
10980   PST,9   MISC = EQ(M,T<L,NO,4>,3)           :F(PST,11)
11000   *
11020   PST,10  OUTPUT = DIFFER(M,T<L,NO,3>,'TR') ' '    :S(PST,11)
11040           MISC = EQ(TRN,SW,OFF)    :S(PST,11)
11060           OUTPUT = EQ(SPRFGOTO,SW,ON) ' '
11080   *
11100   PST,11  MISC = EQ(INSNO,SW,OFF)            :S(PST,12)
11120   *
11140   ** TO IMPLEMENT CONTINUATION THE FOLLOWING LINES WILL
11160   ** HAVE TO BE CHANGED 8975,8990,9020,9030
11180   **
11200           O,LINE = M,T<L,NO,1> M,T<L,NO,5> M,T<L,NO,2> :(PST,13)
11220   *
11240   *
11260   PST,12  O,LINE = M,T<L,NO,5> M,T<L,NO,1> M,T<L,NO,2>
11280   *
11300   PST,13  MISC = DIFFER(M,T<L,NO,3>,'TR') :S(PST,15)
11320   *
11340           MISC = EQ(TRN,SW,OFF)              :S(PST,15)
11360   *
11380     OUTPUT = O.LINE DUPL(' ',100 - SIZE(O.LINE)) '* TRANSFER *' :(PST,17)
11400   *
11420   PST,15  OUTPUT = O,LINE                    :(PST,17)
11440   *
11460   *
11480   PST,17  MISC = IDENT(M,T<L,NO,3>,'IO')  :F(PST,21)
11500           MISC = EQ(FORM,SW,OFF)             :S(PST,21)
11520           NLINE = M,T<L,NO,2>
11540           NLINE BREAK(',') SPAN(',') =
11560            NLINE BREAK(')') , INDEX
11580           INDEX BREAK(',') , INDEX
11600   PST,18  NO = STMNT,NO(INDEX)
11620           NLINE = FORMATS<NO>
11640           NLINE LEN(6) , LABEL
11660           NLINE LABEL =
11680           OUTPUT = LABEL M,T<L,NO,5> NLINE
11700   PST,19  OUTPUT =                           :(PST,25)
11720   PST,21  OUTPUT = EQ(M,T<L,NO,4>,2) ' '   :S(PST,25)
11740           OUTPUT = EQ(M,T<L,NO,4>,3) ' '   :S(PST,25)
11760   *
11780   PST,23  MISC = EQ(M,T<L,NO,4>,20)          :F(PST,25)
11800           MISC = EQ(TRN,SW,OFF)    :S(PST,25)
11820           OUTPUT = DUPL('*',100)
11840           OUTPUT =
11860   *
11880   PST,25             O,K = 5              :(RETURN)
11900   *
11920   *
11940   *
11960   *
11980   *
12000   *************************************************************************************
12020   * THE FOLLOWING ROUTINES THRU 9500 ARE PDP-10/SITBOL SPECIFIC
12040   * AND WERE INCLUDED TO IMPLEMENT SWITCH SELECTION VIA 3 METHODS
12060   *
12080   ***********************************************************
```

```
12100   *
12120   ** REVISE TO ACCEPT ON/OFF OR SINGLE LINE SELECT
12140   **
12160   SWCARD,FILE     TTY = 'ENTER CONTROL SWITCH FILE NAME'
12180           NLINE = TTY
12200           INPUT('INSW',NLINE)
12220           INPUT('INSW','SWCTL,SIT')           :F(ERROR,SWCTLF)
12240           ID,SW = INSW                        :F(ERROR,SWCTLF)
12260           P2DUMP,SW = INSW                    :F(ERROR,SWCTLF)
12280           CGDUMP,SW = INSW                    :F(ERROR,SWCTLF)
12300           CG,MIN = INSW                       :F(ERROR,SWCTLF)
12320           EQ(CG,MIN,0)              :F(SW,1)
12340           CG,MIN = 3
12360   SW,1    CG,MAX = INSW                       :F(ERROR,SWCTLF)
12380           EQ(CG,MAX,0)              :F(SW,2)
12400           CG,MAX = 10
12420   SW,2    CG,SIZE = INSW                      :F(ERROR,SWCTLF)
12440           LT(CG,SIZE,5)             :F(SW,3)
12460           CG,SIZE = 7
12480   SW,3    FORM,SW = INSW                      :F(ERROR,SWCTLF)
12500           DEC,SW = INSW                       :F(ERROR,SWCTLF)
12520           DO,SW = INSW                        :F(ERROR,SWCTLF)
12540           TRN,SW = INSW                       :F(ERROR,SWCTLF)
12560           IO,SW = INSW                        :F(ERROR,SWCTLF)
12580           CG,SW = INSW                        :F(ERROR,SWCTLF)
12600           INSNO,SW = INSW                     :F(ERROR,SWCTLF)
12620           SPBFGOTO.SW = INSW                  :F(ERROR,SWCTLF)
12640           SWCORD,FROM = 'FILE NAMED: ' NLINE
12660           DETACH('INSW')
12680           GT(CG,MAX,B,SIZE)         :F(D,F)
12700   *
12720           OUTPUT = 'ERROR IN CG,MAX, DEFAULT = ' B,SIZE
12740   *
12760           CG,MAX = B,SIZE           :(D,F)
12780   ***************************************************************
12800   ERROR,SWCTLF    TTY = 'ERROR HAS OCCURRED IN SWITCH CONTROL'
12820           TTY = 'FILE READ,FILE DOES NOT EXIST OR AN '
12840           TTY = 'END-OF-FILE WAS ENCOUNTERED DURING SEARCH'
12860           TTY = 'FOR SWITCH CONTROL VALUES, DEFAULT VALUES'
12880           TTY = 'WILL BE USED FOR SWITCHES NOT YET SPECIFIED,'
12900                   TTY =             :(D,F)
12920   ********************************************************************
12940   SWCARD,TTY      TTY = 'ENTER SWITCHES THAT SHOULD BE OFF ='
12960           NLINE = TTY
12980           NLINE 'FORM'      :F(SW,A)
13000           FORM,SW = OFF
13020   SW,A    NLINE 'DEC'       :F(SW,B)
13040           DEC,SW = OFF
13060   SW,B    NLINE 'DO'        :F(SW,C)
13080           DO,SW = OFF
13100   SW,C    NLINE 'TRN'       :F(SW,D)
13120           TRN,SW = OFF
13140   SW,D    NLINE 'IO'        :F(SW,E)
13160           IO,SW = OFF
13180   SW,E    NLINE 'CG'        :F(SW,F)
13200           CG,SW = OFF
13220   SW,F    NLINE 'INSNO'     :F(SW,G)
13240           INSNO,SW = OFF
13260   SW,G    NLINE 'SPBFGOTO'          :F(SW,H)
13280           SPBFGOTO,SW = OFF
```

```
13300   SW.H     NLINE 'ID'         :F(SW,I)
13320            ID,SW = OFF
13340   SW,I     TTY = 'SPECIFY CG, MAXIMUM (5-' B,SIZE ')'
13360            NLINE = TTY
13380            GT(NLINE,B,SIZE)          :S(OUT,BNDS)
13400            LT(NLINE,5)        :S(OUT.BNDS)
13420   SW,J               TTY = 'SPECIFY CG, SIZE (3-9) '
13440            NLINE = TTY
13460            SWCOND.FROM = 'TTY'
13480            GT(NLINE,9)        :S(OUT,BNDS2)
13500            LT(NLINE,3)        :S(OUT,BNDS2)
13520            TTY = 'INPUT OF PARAMETERS COMPLETED' :(D,F)
13540   OUT,BNDS           TTY = 'VALUE OUT OF BOUNDS' :(SW,I)
13560   OUT,BNDS2          TTY = 'VALUE OUT OF BOUNDS' :(SW,J)
13580   ******************************************************
13600   *
13620   *
13640   ******************************************************************************************
13660   ERROR    OUTPUT = 'ERROR'
13680            MISC = DUMP(9)
13700   **************************************************
13720   *
13740   END
```

APPENDIX B

CONCEPTUAL GROUPINGS PROGRAM FOR FORTRAN (GP-F)

6. SOURCE FORTRAN PROGRAM LISTING—— UNGROUPED

PART III --- MAINTENANCE PROGRAMMING

```
*******************************************************
**                                                  **
**                                                  **
** NAME ............................................ **
**                                                  **
** START TIME ....................                   **
**                                                  **
** END TIME   ....................                   **
*******************************************************
```

REVISE THE PAYROLL PROGRAM LISTING FOLLOWING THIS PAGE BY ADDING, INSERTING AND DELETING STATEMENTS TO EFFECT THE FOLLOWING CHANGES:

1. A LISTING IS DESIRED THAT LISTS THE GROSS PAY TOTALS FOR ALL EMPLOYEES BY DEPARTMENTS. THE OUTPUT SHOULD CONSIST OF THE FOLLOWING-

```
DEPT. A        GROSS PAY TOTAL

DEPT. B        GROSS PAY TOTAL

DEPT. C        GROSS PAY TOTAL

ALL DEPTS.     GROSS PAY TOTAL
```

AN EMPLOYEE'S DEPARTMENT MAY BE DETERMINED BY HIS EMPLOYEE NUMBER:

EMPLOYEE NUMBER RANGE	DEPARTMENT
ØØ1 - 199	A
2ØØ - 399	B
4ØØ - 599	C

THE FOLLOWING ARE VARIABLE DESCRIPTIONS WHICH MAY BE HELPFUL-

GP - GROSS PAY FOR THE WEEK

EMP - EMPLOYEE NUMBER FROM MASTER RECORD
EMP2 - EMPLOYEE NUMBER FROM PAYCARD

THE OUTPUT LISTING (GROSS PAY BY DEPARTMENT) SHOULD BE PRINTED ON UNIT NUMBER 7 USING FORMAT STATEMENT NUMBER 777. YOU NEED NOT WRITE THE FORMAT STATEMENT, BUT MAY ASSUME THE FOLLOWING FORMAT STATEMENT WILL BE ADDED TO THE PROGRAM--

```
777 FORMAT(1H1,2ØX,4ØHDEPARTMENT GROSS PAY TABLE BREAKDOWN    //,
   *2ØX,7HDEPT. A,1ØX,F12.2//2ØX,7HDEPT. B,1ØX,F12.2//
   *2ØX,7HDEPT. C,1ØX,F12.2//2ØX,1ØHALL DEPTS.,7,F12.2//)
```

```
C PAYROLL PROGRAM
C
C THE I/O UNITS ARE AS FOLLOWS:
C 1=LINE PRINTER(ERROR MESSAGES)
C 2=OLD MASTER EMPLOYEE PAY RECORD TAPE
C 3=NEW MASTER EMPLOYEE PAY RECORD TAPE
C 4=CHECK PRINTER
C 5=CARD PUNCH FOR NEW PAYROLL/TIME CARDS
C 6=CARD READER FOR EMPLOYEE PAY CARDS
C
      INTEGER EMP,SS1,SS2,SS3,DE1,DE2,DE3,D1,D2,D3,EMP2,V,C
      INTEGER ET1,ET2,ET3,SS11,SS22,SS33
      DIMENSION B(13), T(5), A(5), W(5)
   1Ø READ(2,5,END=9)EMP,SS1,SS2,SS3,(T(J),J=1,5),DE1,DE2,DE3,D1,D2,D3
      READ(2,7,END=9) (B(J),J=1,13)
      IF(D1.EQ.Ø.) GO TO 2Ø
      D=Ø.
      GO TO 15Ø
   2Ø READ(6,22) EMP2,HW,HL,V,C,ET1,ET2,ET3,SS11,SS22,SS33
      READ(6,23) (A(J),J=1,5), (W(J),J=1,5)
      IF(EMP .NE. EMP2 .AND. C .NE. 1.Ø) GO TO 9ØØ
      IF(C.EQ.Ø.) GO TO 4Ø
      IF(C-2.) 25,3Ø,35
   25 EMP=EMP2
      SS1=SS11
      SS2=SS22
      SS3=SS33
      DO 26 J=1,5
   26 T(J)=W(J)
      DE1=ET1
      DE2=ET2
      DE3=ET3
      D1=Ø.
      D2=Ø.
      D3=Ø.
      B(1)=A(1)
      B(2)=Ø.
      B(3)=Ø.
      B(4)=Ø.
      B(5)=Ø.
      B(6)=A(5)
      B(7)=A(3)
      B(8)=Ø.
      B(9)=A(4)
      B(1Ø)=A(2)
      B(11)=Ø.
      B(12)=8Ø.
      B(13)=Ø.
      GO TO 4Ø
   3Ø D1=ET1
      D2=ET2
      D3=ET3
      GO TO 4Ø
   35 IF(A(1).EQ.Ø.) GO TO 36
      B(1)=A(1)
   36 IF(A(5).EQ.Ø.) GO TO 37
      B(6)=A(5)
   37 IF(A(2).EQ.Ø.) GO TO 38
```

```
      B(1Ø)=A(2)
   38 IF(A(3).EQ.Ø.) GO TO 39
      B(7)=A(3)
   39 IF(A(4).EQ.Ø.) GO TO 4Ø
      B(9)=A(4)
   4Ø IF(HW.LE.4Ø.) GO TO 43
      GP=B(1)*4Ø. + (B(1) * 1.5 *(HW-4Ø.))
      GO TO 45
   43 GP=B(1)*HW
   45 IF(V.NE.Ø.) GO TO 5Ø
      B(13)=B(13)+2.
      S=Ø.
      GO TO 9Ø
   5Ø IF(V.EQ.1.) GO TO 6Ø
      B(13)=B(13)-HL
      IF(B(13).GE.Ø.) GO TO 7Ø
      S=(HL-ABS(B(13)))*A(1)
      B(13)=Ø.
      GO TO 9Ø
   6Ø B(12)=B(12)+HL
      IF(B(12).LE.8Ø.) GO TO 7Ø
      D=B(12)-8Ø.
      G=HL-D
      S=G*A(1)
      B(12)=8Ø.
      GO TO 9Ø
   7Ø S=HL*A(1)
   9Ø GP=S+GP
      FINC=(GP-(13.*B(6)))*.14
      B(3)=B(3)+FINC
  1ØØ B(2)=GP+B(2)
      IF(B(2).LE.48ØØ.) GO TO 1Ø5
      FICA=Ø.
      GO TO 11Ø
  1Ø5 FICA=GP*.Ø3625
  11Ø B(4)=FICA+B(4)
      IF(B(2).LE.51ØØ.) GO TO 115
      SDI=Ø.
      GO TO 12Ø
  115 SDI=.Ø1*GP
  12Ø B(5)=SDI+B(5)
      IF(B(5).LE.51.) GO TO 125
      B(5)=51.
  125 B(11)=B(11)+B(1Ø)
      IF(B(7).EQ.Ø.) GO TO 135
      B(8)=B(6)+B(7)
      BOND=.75*B(9)
      IF(BOND.LT.B(8)) GO TO 13Ø
      GO TO 135
  13Ø B(8)=B(8)-BOND
      WBOND=B(9)
      GO TO 14Ø
  135 WBOND=Ø.
  14Ø D=(GP-(FINC+FICA+SDI+B(1Ø)))+WBOND
  15Ø WRITE(3,5) EMP,SS1,SS2,SS3,(T(J),J=1,5),DE1,DE2,DE3,D1,D2,D3
      WRITE(3,7) (B(J),J=1,13)
      WRITE(5,17) EMP,SS1,SS2,SS3,(T(J),J=1,5)
      WRITE(4,18) (T(J),J=1,5),SS1,SS2,SS3,D
      IF(C .EQ. 1.Ø) GO TO 2Ø
      GO TO 1Ø
```

```
 900  WRITE(1,901) EMP,(T(J),J=1,5),EMP2,(W(J),J=1,5)
   9  STOP
   7  FORMAT(13F6.2)
  22  FORMAT(I4,2F4.2,2I1,3I2,I3,I2,I4)
  23  FORMAT(5F6.2,5X,5A5)
  17  FORMAT(I4,16X,I3,I2,I4/35X,5A5)
  18  FORMAT(5A5,3I4,F8.2)
   5  FORMAT(4I4,5A5,6I2)
 901  FORMAT(10H ERROR---  ,I4,2X,5A5,4X,I4,2X,5A5)
      END
```

```
EOF ON DSKA0:ICS.1
```

APPENDIX B

CONCEPTUAL GROUPINGS PROGRAM FOR FORTRAN (GP-F)

7. SOURCE FORTRAN PROGRAM LISTING—GROUPED

PART III --- MAINTENANCE PROGRAMMING

```
**************************************************
**                                              **
**                                              **
** NAME .................................       **
**                                              **
** START TIME ...................               **
**                                              **
** END TIME   ...................               **
**************************************************
```

REVISE THE PAYROLL PROGRAM LISTING FOLLOWING THIS PAGE BY ADDING, INSERTING AND DELETING STATEMENTS TO EFFECT THE FOLLOWING CHANGES:

1. A LISTING IS DESIRED THAT LISTS THE GROSS PAY TOTALS FOR ALL EMPLOYEES BY DEPARTMENTS. THE OUTPUT SHOULD CONSIST OF THE FOLLOWING-

```
DEPT. A      GROSS PAY TOTAL

DEPT. B      GROSS PAY TOTAL

DEPT. C      GROSS PAY TOTAL

ALL DEPTS.   GROSS PAY TOTAL
```

AN EMPLOYEE'S DEPARTMENT MAY BE DETERMINED BY HIS EMPLOYEE NUMBER:

EMPLOYEE NUMBER RANGE	DEPARTMENT
001 - 199	A
200 - 399	B
400 - 599	C

THE FOLLOWING ARE VARIABLE DESCRIPTIONS WHICH MAY BE HELPFUL-

GP - GROSS PAY FOR THE WEEK

EMP - EMPLOYEE NUMBER FROM MASTER RECORD
EMP2 - EMPLOYEE NUMBER FROM PAYCARD

THE OUTPUT LISTING (GROSS PAY BY DEPARTMENT) SHOULD BE PRINTED ON UNIT NUMBER 7 USING FORMAT STATEMENT NUMBER 777. YOU NEED NOT WRITE THE FORMAT STATEMENT, BUT MAY ASSUME THE FOLLOWING FORMAT STATEMENT WILL BE ADDED TO THE PROGRAM--

```
777 FORMAT(1H1,20X,40HDEPARTMENT GROSS PAY TABLE BREAKDOWN   //,
   *20X,7HDEPT. A,10X,F12.2//20X,7HDEPT. B,10X,F12.2//
   *20X,7HDEPT. C,10X,F12.2//20X,10HALL DEPTS.,7,F12.2//)
```

```
      INTEGER EMP,SS1,SS2,SS3,DE1,DE2,DE3,D1,D2,D3,EMP2,V,C
      INTEGER ET1,ET2,ET3,SS11,SS22,SS33
      DIMENSION B(13), T(5), A(5), W(5)

C PAYROLL PROGRAM
C
C THE I/O UNITS ARE AS FOLLOWS:
C 1=LINE PRINTER(ERROR MESSAGES)
C 2=OLD MASTER EMPLOYEE PAY RECORD TAPE
C 3= NEW MASTER EMPLOYEE PAY RECORD TAPE
C 4=CHECK PRINTER
C 5=CARD PUNCH FOR NEW PAYROLL/TIME CARDS
C 6=CARD READER FOR EMPLOYEE PAY CARDS
C

   10 READ(2,5,END=9)EMP,SS1,SS2,SS3,(T(J),J=1,5),DE1,DE2,DE3,D1,D2,D3
    5 FORMAT(4I4,5A5,6I2)

      READ(2,7,END=9) (B(J),J=1,13)
    7 FORMAT(13F6.2)

      IF(D1.EQ.Ø.) GO TO 2Ø                                                            * TRANSFER *

      D=Ø.
      GO TO 15Ø                                                                        * TRANSFER *
----------------------------------------------------------------------------------------

   2Ø READ(6,22) EMP2,HW,HL,V,C,ET1,ET2,ET3,SS11,SS22,SS33
   22 FORMAT(I4,2F4.2,2I1,3I2,I3,I2,I4)

      READ(6,23) (A(J),J=1,5), (W(J),J=1,5)
   23 FORMAT(5F6.2,5X,5A5)

      IF(EMP .NE. EMP2 .AND. C .NE. 1.Ø) GO TO 9ØØ                                     * TRANSFER *

      IF(C,EQ.Ø,) GO TO 4Ø                                                             * TRANSFER *

      IF(C-2,) 25,3Ø,35                                                                * TRANSFER *

   25 EMP=EMP2
      SS1=SS11
      SS2=SS22
      SS3=SS33

            DO 26 J=1,5

   26       T(J)=W(J)
            DE1=ET1
            DE2=ET2
            DE3=ET3
            D1=Ø.
            D2=Ø.
            D3=Ø.
            B(1)=A(1)
```

```
              B(2)=Ø.
              B(3)=Ø.
   --------------------------------------------------------------------

              B(4)=Ø.
              B(5)=Ø.
              B(6)=A(5)
              B(7)=A(3)
              B(8)=Ø.
              B(9)=A(4)
              B(1Ø)=A(2)
              B(11)=Ø.
   --------------------------------------------------------------------

              B(12)=8Ø.
              B(13)=Ø.
              GO TO 4Ø                                                          * TRANSFER *
   --------------------------------------------------------------------

     3Ø       D1=ET1
              D2=ET2
              D3=ET3
              GO TO 4Ø                                                          * TRANSFER *
   --------------------------------------------------------------------

     35       IF(A(1).EQ.Ø.) GO TO 36                                           * TRANSFER *

              B(1)=A(1)
     36       IF(A(5).EQ.Ø.) GO TO 37                                           * TRANSFER *

              B(6)=A(5)
     37       IF(A(2).EQ.Ø.) GO TO 38                                           * TRANSFER *

              B(1Ø)=A(2)
     38       IF(A(3).EQ.Ø.) GO TO 39                                           * TRANSFER *

              B(7)=A(3)
     39       IF(A(4).EQ.Ø.) GO TO 4Ø                                           * TRANSFER *

              B(9)=A(4)
     4Ø       IF(HW.LE.4Ø.) GO TO 43                                            * TRANSFER *

              Gp=B(1)*4Ø. + (B(1) * 1.5 *(HW-4Ø.))
              GO TO 45                                                          * TRANSFER *
   --------------------------------------------------------------------

     43       GP=B(1)*HW
     45       IF(V.NE.Ø.) GO TO 5Ø                                              * TRANSFER *

              B(13)=B(13)+2.
              S=Ø.
              GO TO 9Ø                                                          * TRANSFER *
   --------------------------------------------------------------------

     5Ø       IF(V.EQ.1.) GO TO 6Ø                                              * TRANSFER *

              B(13)=B(13)-HL
              IF(B(13).GE.Ø.) GO TO 7Ø                                          * TRANSFER *

              S=(HL-ABS(B(13)))*A(1)
```

```
             B(13)=0.
             GO TO 90                                                    * TRANSFER *
      ------------------------------------------------------------------

  60         B(12)=B(12)+HL
             IF(B(12).LE.80.) GO TO 70                                   * TRANSFER *

             D=B(12)-80.
             G=HL-D
             S=G*A(1)
             B(12)=80.
             GO TO 90                                                    * TRANSFER *
      ------------------------------------------------------------------

  70         S=HL*A(1)
  90         GP=S+GP
             FINC=(GP-(13.*B(6)))*.14
             B(3)=B(3)+FINC
 100         B(2)=GP+B(2)
             IF(B(2).LE.4800.) GO TO 105                                 * TRANSFER *

             FICA=0.
             GO TO 110                                                   * TRANSFER *
      ------------------------------------------------------------------

 105         FICA=GP*.03625
 110         B(4)=FICA+B(4)
             IF(B(2).LE.5100.) GO TO 115                                 * TRANSFER *

             SDI=0.
             GO TO 120                                                   * TRANSFER *
      ------------------------------------------------------------------

 115         SDI=.01*GP
 120         B(5)=SDI+B(5)
             IF(B(5).LE.51.) GO TO 125                                   * TRANSFER *

             B(5)=51.
 125         B(11)=B(11)+B(10)
             IF(B(7).EQ.0.) GO TO 135                                    * TRANSFER *

             B(8)=B(8)+B(7)
             BOND=.75*B(9)
             IF(BOND.LT.B(8)) GO TO 130                                  * TRANSFER *

             GO TO 135                                                   * TRANSFER *
      ------------------------------------------------------------------

 130         B(8)=B(8)-BOND
             WBOND=B(9)
             GO TO 140                                                   * TRANSFER *
      ------------------------------------------------------------------

 135         WBOND=0.
 140         D=(GP-(FINC+FICA+SDI+B(10)))+WBOND

 150         WRITE(3,5) EMP,SS1,SS2,SS3,(T(J),J=1,5),DE1,DE2,DE3,D1,D2,D3
   5         FORMAT(4I4,5A5,6I2)
```

```
              WRITE(3,7) (F(J),J=1,13)
    7         FORMAT(13F6.2)

              WRITE(5,17) EMP,SS1,SS2,SS3,(T(J),J=1,5)
   17         FORMAT(I4,16X,I3,I2,I4/35X,5A5)

              WRITE(4,18) (T(J),J=1,5),SS1,SS2,SS3,D
   18         FORMAT(5A5,3I4,F6.2)

              IF(C .EQ. 1.Ø) GO TO 2Ø                                                   * TRANSFER *

              GO TO 1Ø                                                                  * TRANSFER *
  -----------------------------------------------------------------------------------


  9ØØ         WRITE(1,9Ø1) EMP,(T(J),J=1,5),EMP2,(W(J),J=1,5)
  9Ø1         FORMAT(1ØH ERROR--- ,I4,2X,5A5,4X,I4,2X,5A5)

    9         STOP
              END
```

```
EOF ON DSKAØ:ICS.3
```

DEVELOPMENTS IN COMPUTER AIDED
SOFTWARE MAINTENANCE

DCASM Final Report

APPENDIX C

CONCEPTUAL GROUPINGS PROGRAM FOR PL/1 (GP-P)

(Prepared under Contract F19628-74-C-0061 by AMS, Inc.,
401 N. Harvard Avenue, Claremont, California 91711.)

DEVELOPMENTS IN COMPUTER AIDED
SOFTWARE MAINTENANCE

DCASM Final Report

APPENDIX C

Table of Contents

APPENDIX C

1. General System Description (GP-P)

Part I

This version of GP-P allots 50 buffer lines in all for statement storage. LINDEX, the table of pointers to first lines of statements in the buffer, is dimensioned at 20. ADDPTR, which computes the address of the next available buffer line, checks for overwritten storage.

The buffer is actually 6 arrays, of which 2 are doubly dimensioned:

PREFIX (50,5) contains any condition prefixes, each < = 31 characters, which precede the statement.

LABEL (50,5) stores any statement labels (also < = 31 characters each)

TYPE (50) contains the classification: 'AST' for assignment, IO, 'CAL' for call statements, 'CTRL' for GO TO, WAIT, RETURN and other control commands, 'STOP' for declarations and allocations, 'ON' for 'ON CONDITION' statements, and 'COM' for comments.

LEVEL (50) holds the nesting level. It is always 0 for a comment, 1 for the main program. Levels presently range from 0 to 9.

SKIP (50) contains the SKIP code assigned to the statement in Part II. This code indicates lines to be skipped and/or dotted lines to be used in setting the statement in its proper format. In Part III the code may be separated into its 'fore' and 'aft' components, or may be modified, when more than one statement at a time is considered. The doubly dimensioned CHCODE arrays contains these 2 components for each value of SKIP.

SKIP	Action to be taken	CHCODE (SKIP, 1) (before)	CHCODE (SKIP, 2) (after)
0	no skip before or after	0	0
1	skip line before	1	0
2	skip line before & after	1	3
3	skip line after	0	3
4	dotted line after	0	4
5	dotted line & skip, after	0	5
6	precede by dotted line & skip	6	0
7	precede by dot & skip, follow by dot	6	4
8	precede by dotted line	8	0
9	precede & follow by dotted line	8	4

TEXT (50) stores the statement text after any condition and/or label prefixes have been stored separately. It allows up to 120 characters to the line (but uses less for 80 column output).

The pushdown list is a set of 3 controlled variables, STACK 1, STACK 2 and STACK 3 (STACK 2 is dimensioned at 5) which are allocated and freed as levels increase or decrease. The same list serves to keep track of commands, labels and levels for IF...ELSE structures and for all kinds of blocks.

Part II

The program processes one (1) PL/1 statement at a time. The statement is read, analyzed, and stored with its: (1) type; (2) skip code; (3) nesting level; (4) text; and (5) prefix(es) which are separated.

Part III

After each statement is analyzed and stored (Part II), Part III surveys the situation to see if any output is to take place, to output if so indicated, and to reindex the statements remaining in the buffer. When COUNT has reached the stipulated size BUFCG, the buffer is examined to find conceptual groups (CGs), and these are output if found. If not, the top half of the buffer is output and the program loops with the next statement.

Rule 1

Every time the level changes that part of the buffer preceding the change is output.

Rule 2

If the SKIP code of the current statement calls for skip and/or dotted line before it, attach the 'action' (as an 'after' skip or dotted line) to the statement preceding it, and after the current statement's code to show only its 'after' component. (Note that when this rule is applied to every statement in turn, the code of the statement preceding the current one can be only 0, 3, 4 or 5).

Rule 3

If the SKIP code of statement preceding the current one calls for skip and/or dotted line after it, output all lines in buffer through that preceding statement with that code. Again, the code can be only 0, 3, 4 or 5.

Note that in Rules 2 and 3, we consider the 'before' code of the current statement, and the 'after' code of its predicessor, and the current statement is never printed out (unless it is the last statement in the program, Rule 6). This is because a succeeding statement of the same type may erase the 'skip line' following a statement. Consequently the COUNT is never less than 1 after execution starts. Also, none but the current statement and its predecessor can have a SKIP code other than 0, else it would have been output.

Rule 4 (Search for CG)

When COUNT reaches BUFCG, a search is made for a CG. If in any group of HALFCG (half of BUFCG) successive statements at least TESTCG are of the same type, it is considered a CG. Only assignment, IO and CALL statements are counted, since other types carry punctuation codes which would have output them before this. Each group of HALFCG is considered in turn, starting with statement 1 the first time, statement 2 the next, etc. until the next-to-last buffer statement (the last, if it is EOF), has been examined. BUFCG and TESTCG are read-in values. BUFCG cannot exceed 20 without changing the dimensions given in Part I declarations, and TESTCG must obviously not exceed half of BUFCG.

If a CG is found, the pre-CG statements are output (no pre-skip), a skip is printed, the CG is output followed by a skip, and the remaining statements are 'moved up' in the buffer.

Rule 5

If no CG is found, the top HALFCG of the buffer is output without any skips, and remaining statements are 'moved up' in buffer.

Rule 6

If an EOF has occurred (FINIS = 1), the whole buffer is output if COUNT < HALFCG. If COUNT is not less than HALFCG, a search for a CG is made, using all the statements in the buffer. In either case, all statements are output, and the program proceeds to ENDPROG.

Note(s):

STEXT (SOMLIN)

Before STEXT is called, the calling program must have defined TYPE, SKIP code and LEVEL for the statement. The condition prefix(es) and/or label(s), if any, have already been stored in the buffer among PREFIX and LABEL. (Up to 5 each, of ≤ characters, are allowed here.) The statement text is in SOMLIN. STEXT must now format the text for output.

FORMAT: In all cases column 1 is blank. If the statement is a comment, cols 2-4 contain the characters '/*b̸' followed by 65 characters of text if NCOL = 80, by 109 if NCOL = 120. These are followed by 'b̸*/' in cols 70-72 (or 114-116).

For non-comment statements the maximum length for an output line (excluding the level) is 72-Margin characters (116-margin for 120 column output), where Margin is the margin defined for the nesting level of this statement (stored in LEVEL). The level is in cols 74 & 75 (118 & 119).

If the statement occupies more than one line, the SKIP code is separated into 'fore' and 'aft' components (using the CHCODE table), the 'fore' code is assigned to the first line, the 'aft' to the last, with any additional lines between marked 0, or no skip.

If condition or label prefixes are present they will be printed on the same line as the text only if their (combined) length fits into the margin before the text. Otherwise they will be assigned separate line(s).

CONCEPTUAL GROUPINGS PROGRAM FOR PL/1

2. Operating Instructions

(1) The Conceptual Grouping Program for PL/1 has been designed to operate under the standard operating procedures for IBM 360/370 (DS or DOS) systems.

(2) Input source deck is read off SYSIPT (device independent).

(3) Control card information is read off SYSØØY (a 2540 card reader), and it consists of 7 numbers separated by one or more spaces on one or more cards:

Parameter 1: Length of printline; suggested 120.
2: Beginning margin; must be at least 9, suggested 9.
3: Margin step size; suggested 5.
4: The number 20.
5: The number 7.
6: Max. no. of characters per PL/1 statement; suggested 800.
7: Max. no. of edited lines produced per PL/1 statement; suggest 50.

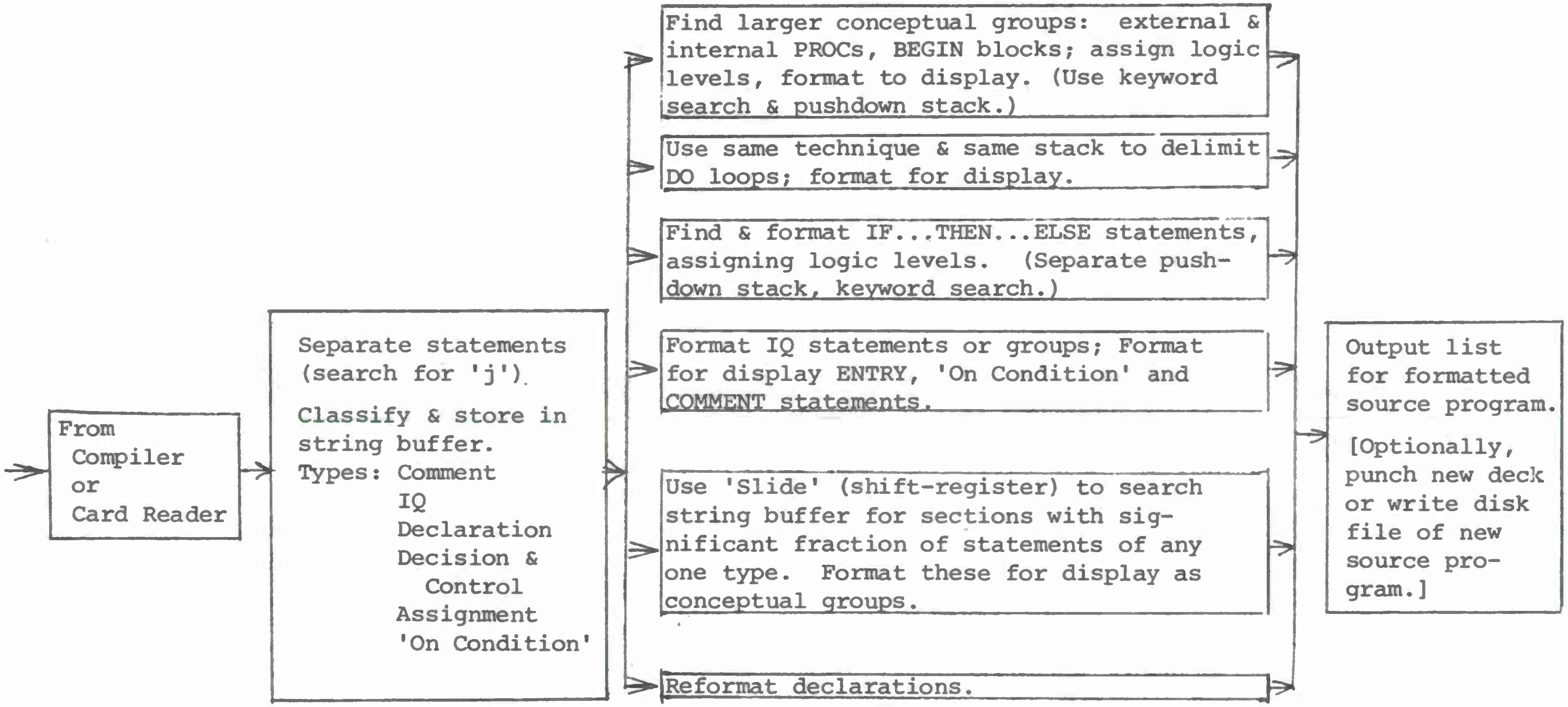
(4) Parameters 6, 7 will affect storage requirements. If parameter 6 is too small, program will know this and cancel after printing an appropriate error message.

(5) Parameter 7, however, will not be recognized as being too small if, indeed, it is too small. Missing edited lines are an indication that parameter 7 was too small.

APPENDIX C

CONCEPTUAL GROUPINGS PROGRAM FOR PL/1 (GP-P)

3. GROUPINGS PROGRAM FOR PL/1 SYSTEM BLOCK DIAGRAM



From Compiler or Card Reader

Separate statements (search for 'j').

Classify & store in string buffer.
Types: Comment
IQ
Declaration
Decision & Control
Assignment
'On Condition'

Find larger conceptual groups: external & internal PROCs, BEGIN blocks; assign logic levels, format to display. (Use keyword search & pushdown stack.)

Use same technique & same stack to delimit DO loops; format for display.

Find & format IF...THEN...ELSE statements, assigning logic levels. (Separate pushdown stack, keyword search.)

Format IQ statements or groups; Format for display ENTRY, 'On Condition' and COMMENT statements.

Use 'Slide' (shift-register) to search string buffer for sections with significant fraction of statements of any one type. Format these for display as conceptual groups.

Reformat declarations.

Output list for formatted source program.

[Optionally, punch new deck or write disk file of new source program.]

GP-P SYSTEM BLOCK DIAGRAM

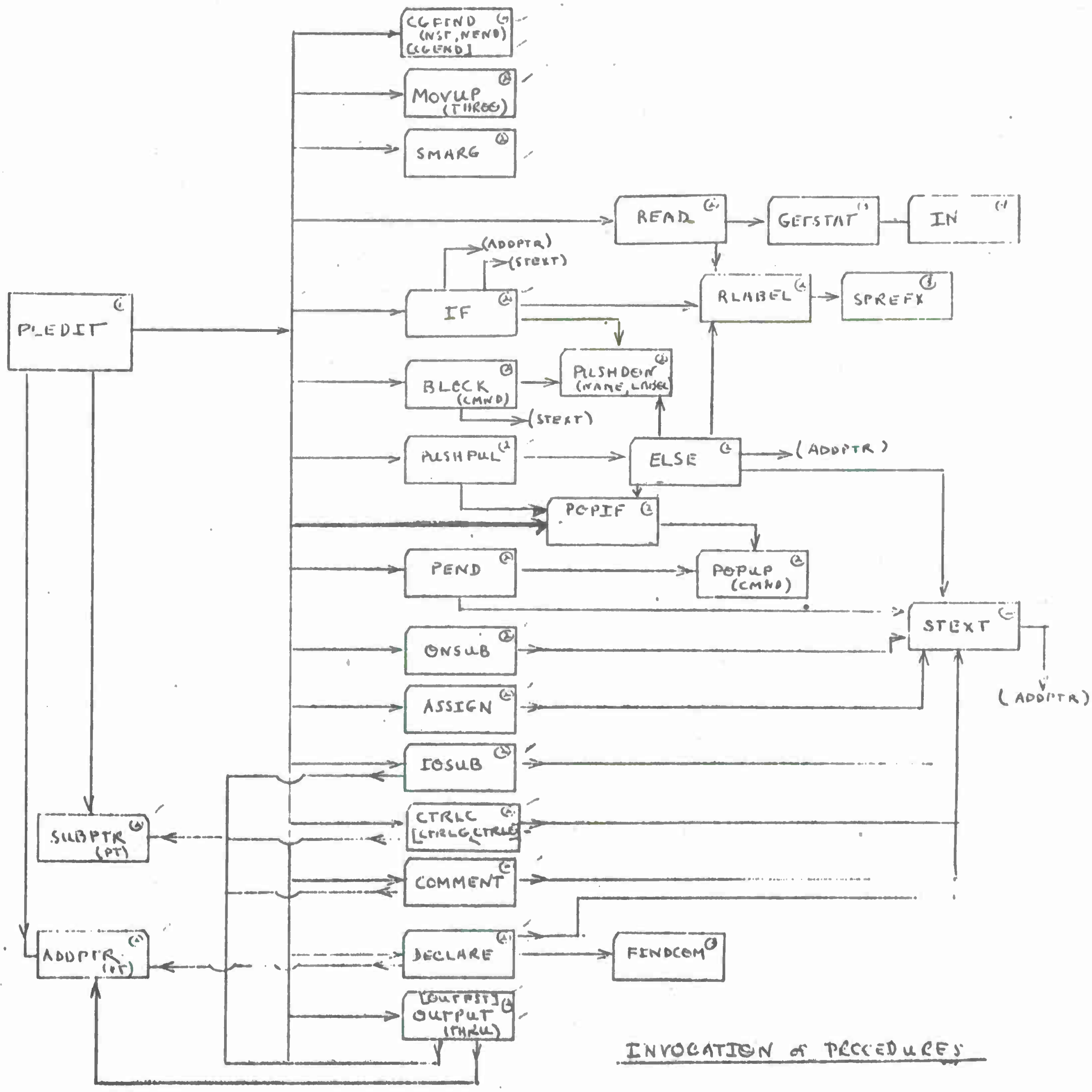
GROUPING PROGRAM FOR PL/1

APPENDIX C

CONCEPTUAL GROUPINGS PROGRAM FOR PL/1 (GP-P)

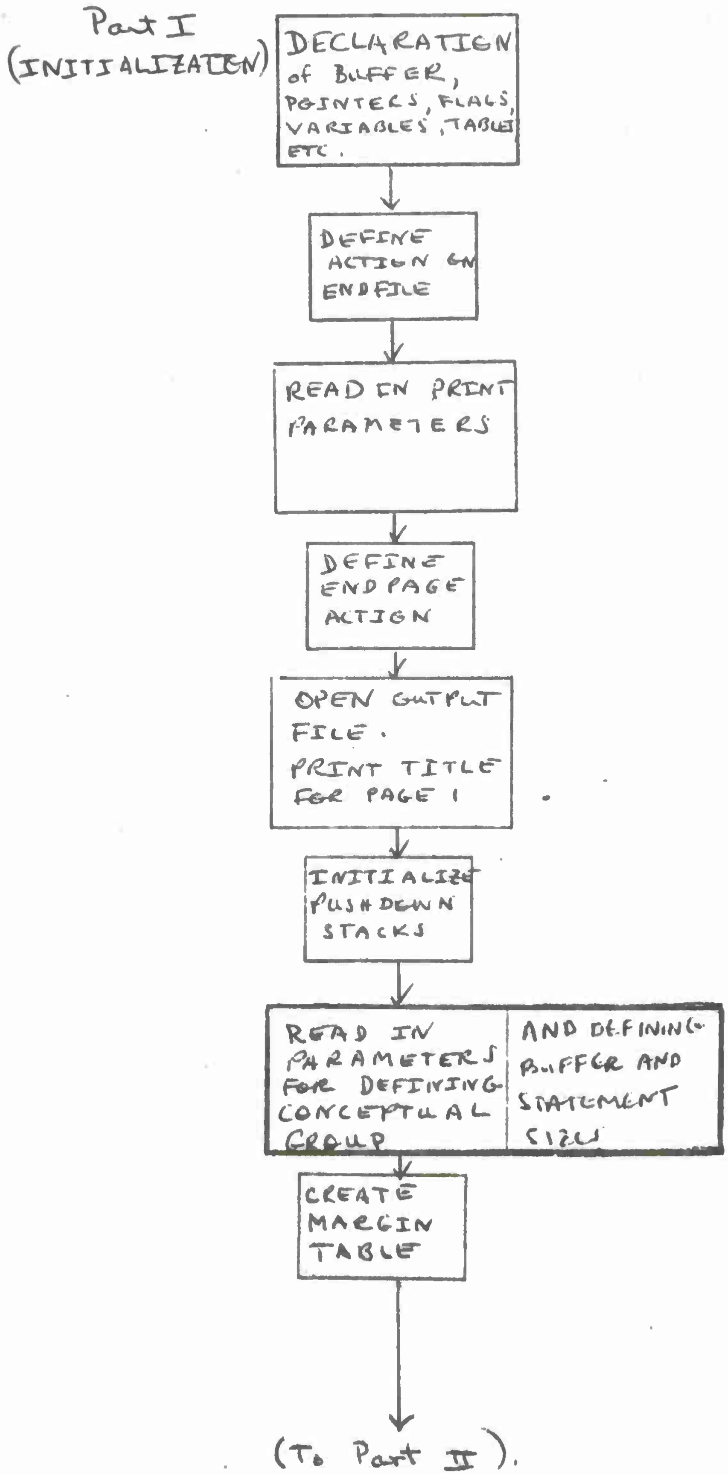
4. GROUPINGS PROGRAM FOR PL/1 (GP-P)

GP-P FLOW CHARTS

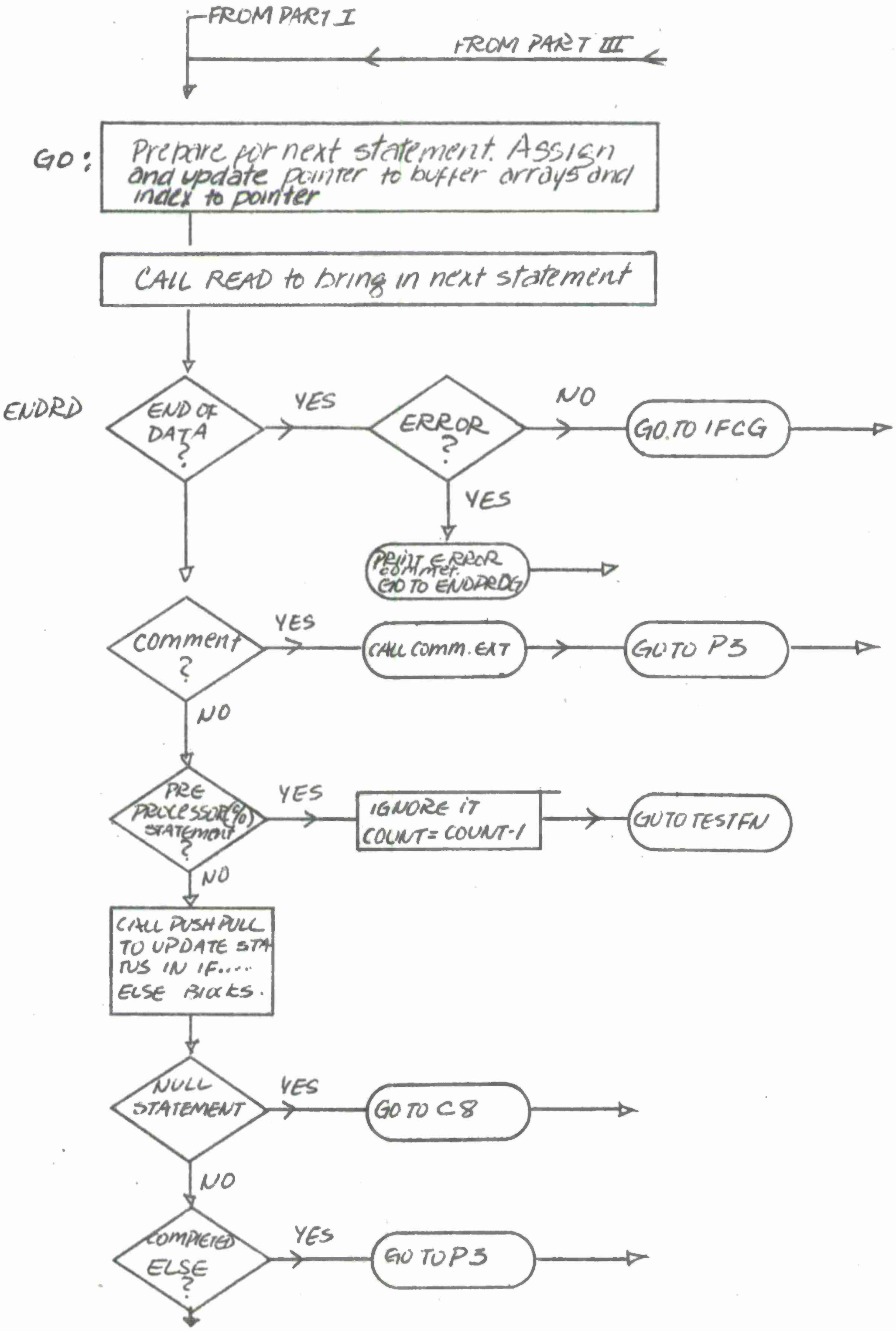


INVOCATION of PROCEDURES

PLEDIT:

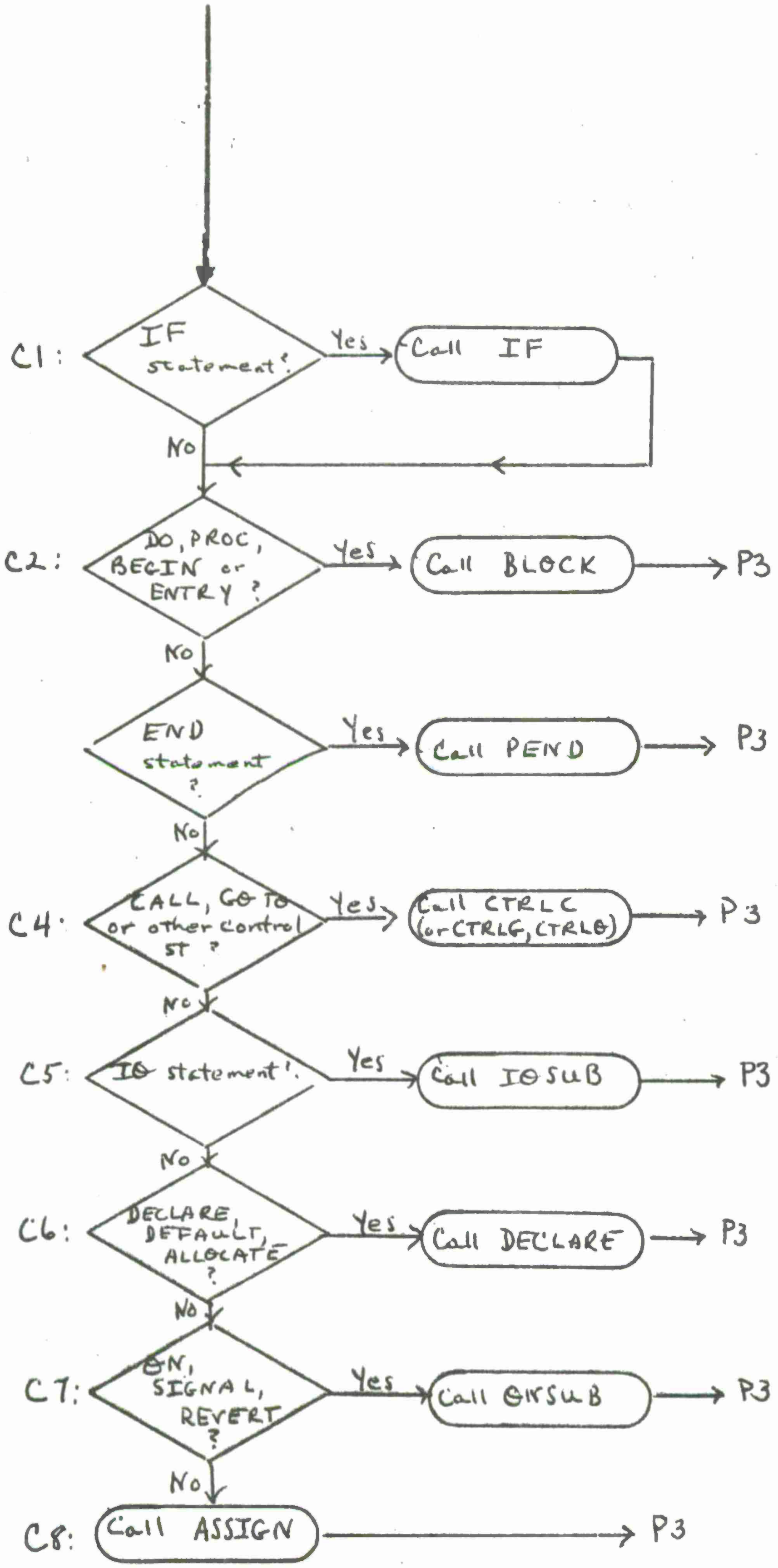

Part I
(INITIALIZATION)
DECLARATION of BUFFER, POINTERS, FLAGS, VARIABLES, TABLES ETC.
DEFINE ACTION ON ENDFILE
READ IN PRINT PARAMETERS
DEFINE ENDPAGE ACTION
OPEN OUTPUT FILE.
PRINT TITLE FOR PAGE 1
INITIALIZE PUSHDOWN STACKS
READ IN PARAMETERS FOR DEFINING CONCEPTUAL GROUP
AND DEFINING BUFFER AND STATEMENT SIZES
CREATE MARGIN TABLE
(To Part II).

PLEDIT (cont.): PART II Process 1 statement at a time, Read, analyze, store with its type, skip code, nesting level, text, and (seperated) prefix(es)


FROM PART I
FROM PART III
GO:
Prepare for next statement. Assign and update pointer to buffer arrays and index to pointer
CALL READ to bring in next statement
ENDRD
END OF DATA ?
YES
ERROR ?
NO
GO TO IFCG
YES
PRINT ERROR COMMET. GO TO ENDPRDG
comment ?
YES
CALL COMM.EXT
GO TO P3
NO
PRE PROCESSOR(%) STATEMENT ?
YES
IGNORE IT COUNT=COUNT-1
GO TO TESTFN
NO
CALL PUSHPULL TO UPDATE STATUS IN IF.... ELSE BLOCKS.
NULL STATEMENT
YES
GO TO C8
NO
COMPLETED ELSE ?
YES
GO TO P3

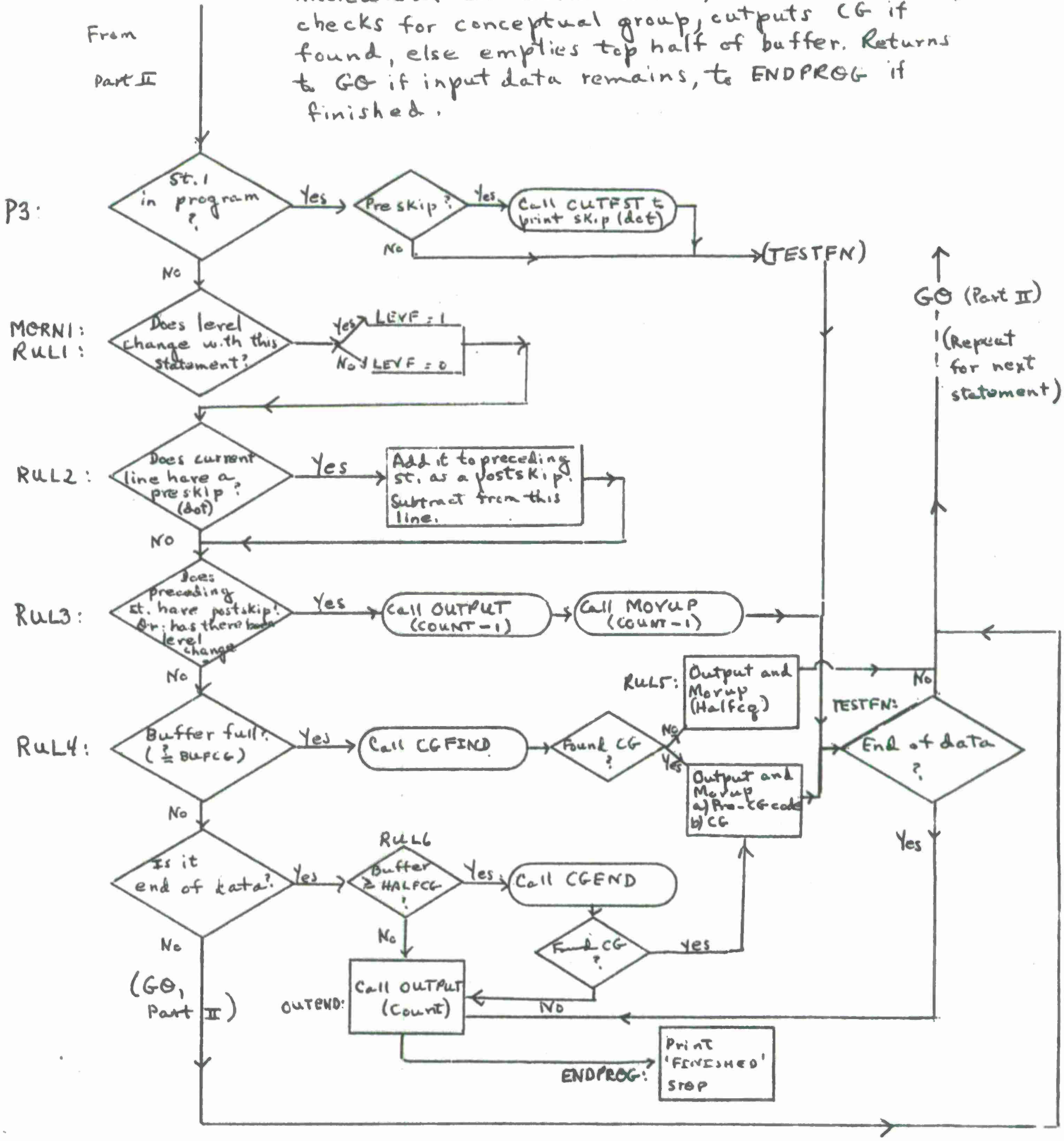
PLEDIT (Cont.)

Part II (Cont)



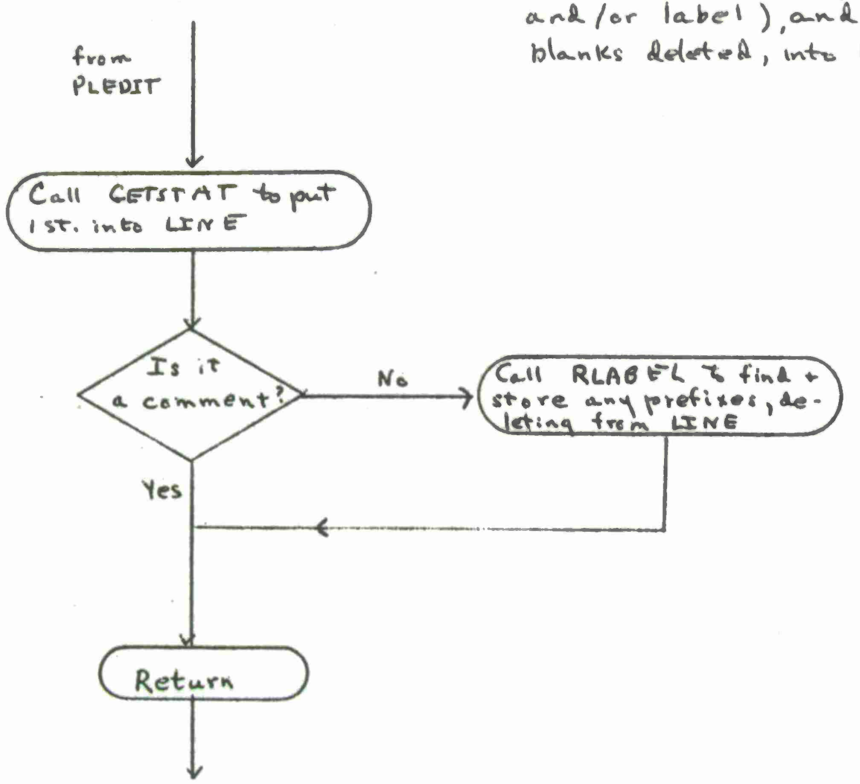
PLEDIT (Cont)

Part III: On return from type subroutine(s) Part III checks for output and performs output if indicated. If buffer is full, or if end of data, checks for conceptual group, outputs CG if found, else empties top half of buffer. Returns to GO if input data remains, to ENDPROG if finished.



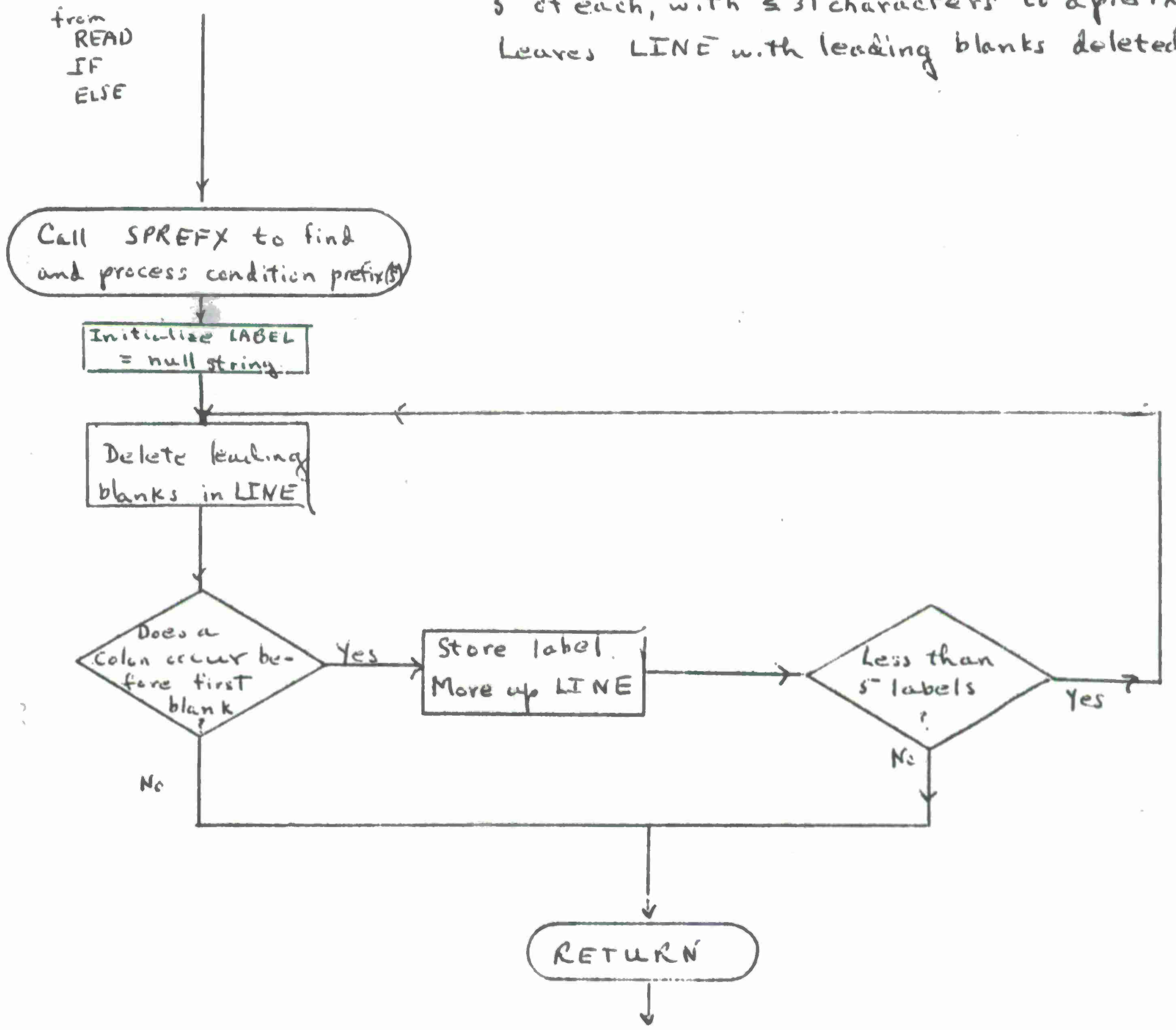
READ (Subroutine)

Reads input file to separate one statement, stores prefix(es), (condition and/or label), and puts text, leading blanks deleted, into LINE.

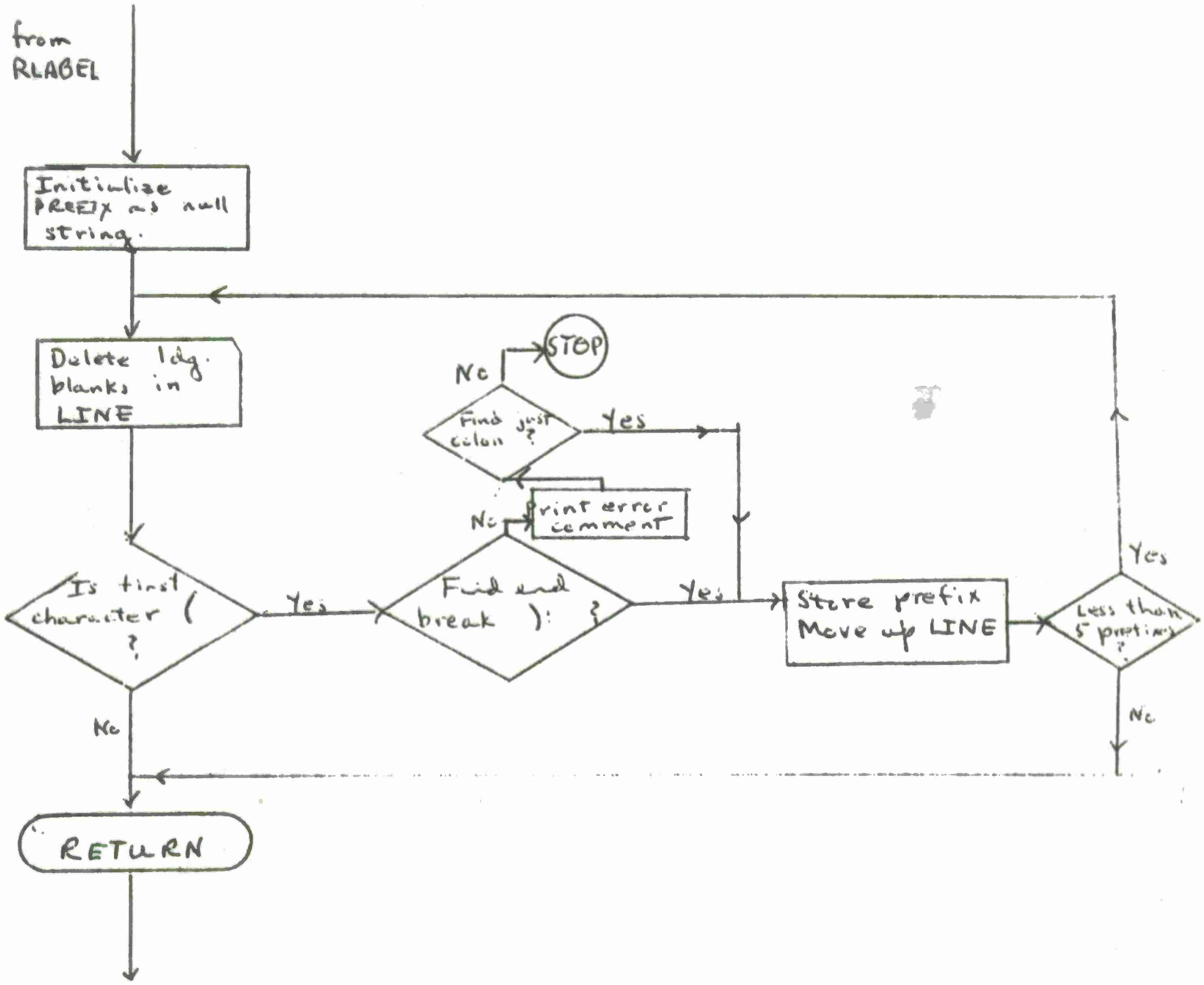

from PLEDIT
Call GETSTAT to put 1st. into LINE
Is it a comment?
No
Call RLABEL to find + store any prefixes, deleting from LINE
Yes
Return

RLABEL

Called by READ, IF or ELSE to find, store and delete from LINE any condition or label prefixes. Allows up to 5 of each, with ≤ 31 characters to a prefix, Leaves LINE with leading blanks deleted.


from
READ
IF
ELSE
Call SPREFX to find and process condition prefix(s)
Initialize LABEL = null string
Delete leading blanks in LINE
Does a Colon occur before first blank ?
Yes
No
Store label Move up LINE
Less than 5 labels ?
Yes
No
RETURN

SPREFX


from RLABEL
Initialize PREFIX as null string.
Delete ldg. blanks in LINE
Is first character (?
Yes
No
Find end break): ?
Yes
No
Print error comment
Find just colon ?
Yes
No
STOP
Store prefix Move up LINE
Less than 5 prefixes ?
Yes
No
RETURN

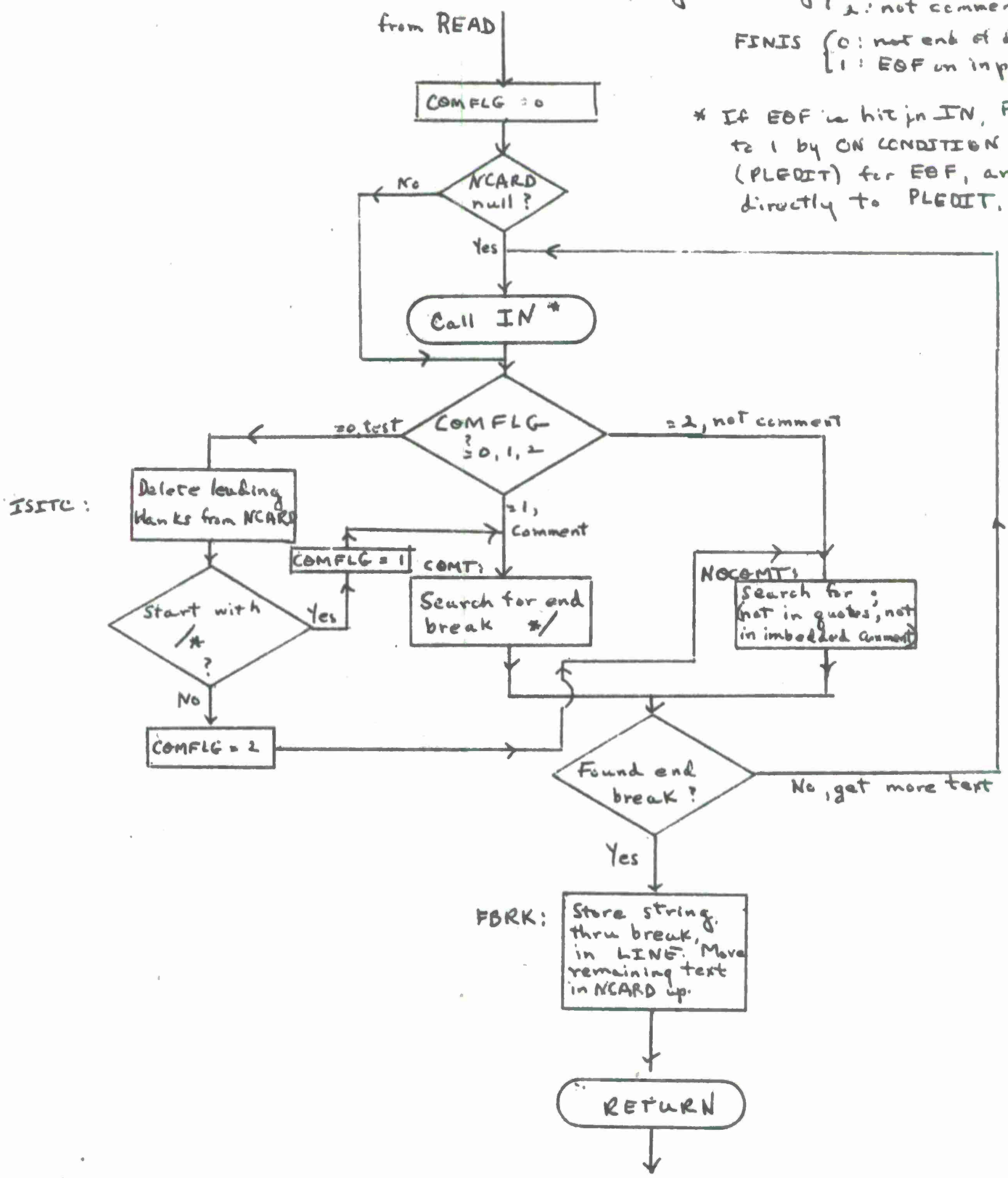
GETSTAT

Called by READ to put 1 statement, leading + trailing blanks deleted, into LINE. Calls IN to input 1 record into NCARD.

Flags: Comflg { 0 : test for comment
1 : comment
2 : not comment

FINIS { 0 : not end of data
1 : EOF on input

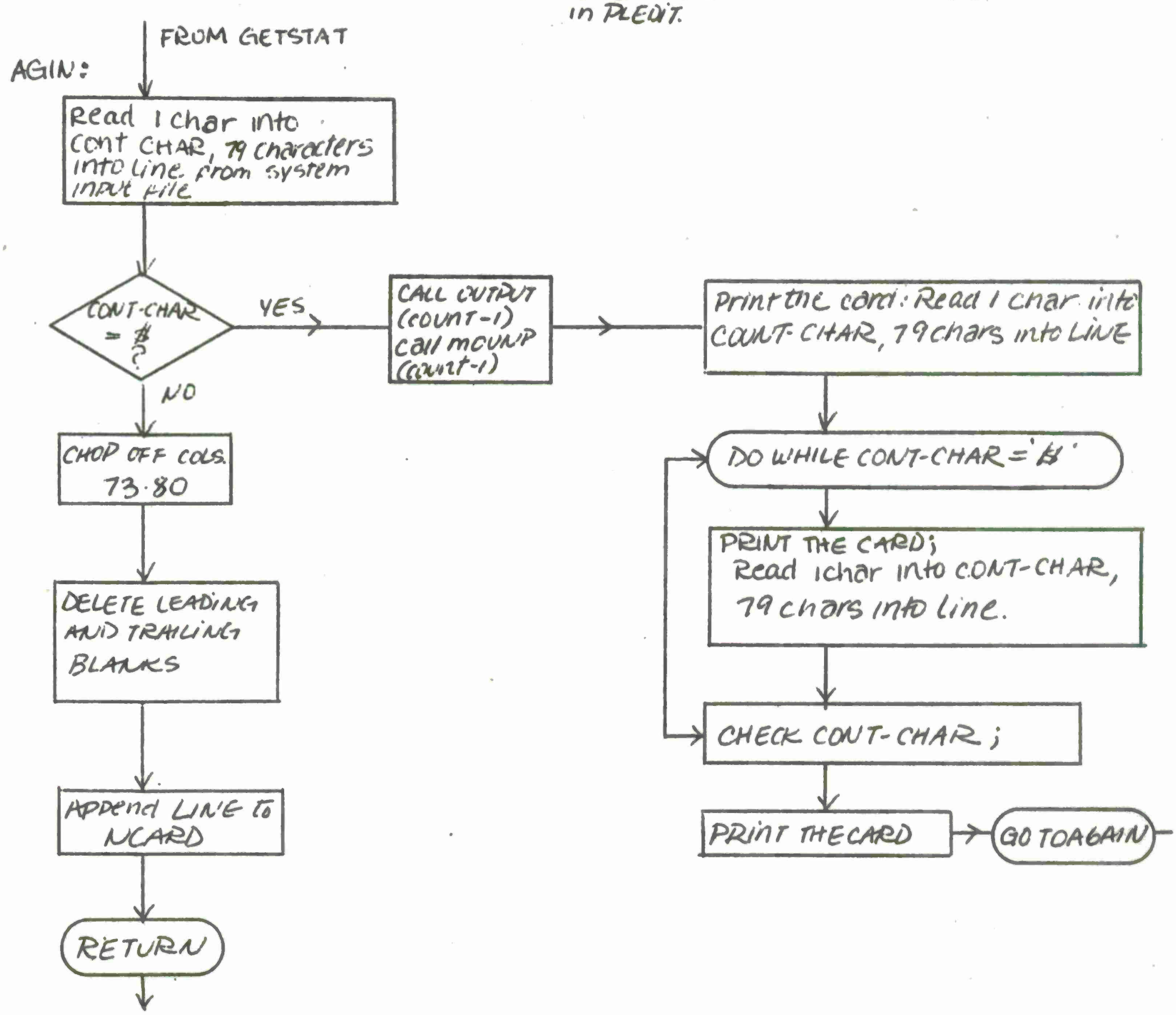
* If EOF is hit in IN, FINIS is set to 1 by ON CONDITION block defined (PLEDIT) for EOF, and IN returns directly to PLEDIT.



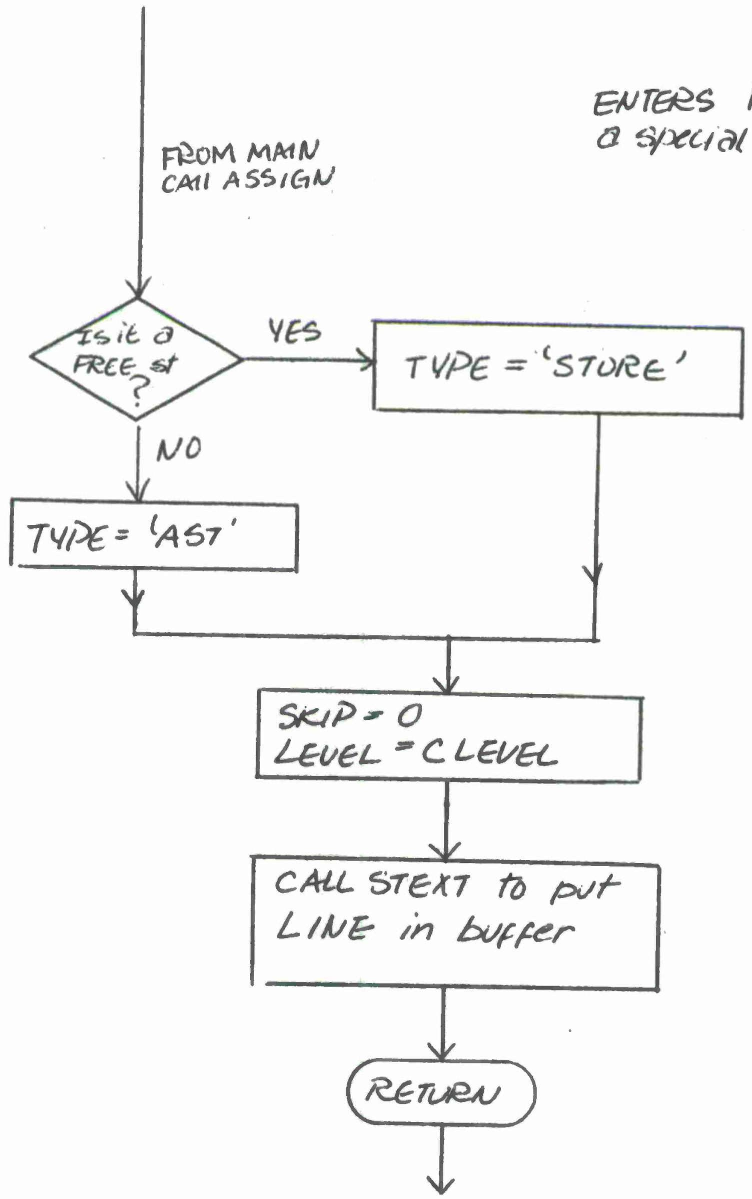
IN

Called by GETSAT to fetch 1 input record, delete leading and trailing blanks, and add it to NCARD. Written to read 80-column input with data in cols. 2-72. If '$' detected in col. 1, that card up to next card with '$' in col. 1 (inclusive) are printed but not processed after OUTPUT, MOVUP called.

* Action on end of input file is defined by ON END FILE block in PLEDIT: Finis=1 and return is made to ENDRD in PLEDIT.


FROM GETSTAT
AGIN:
Read 1 char into cont CHAR, 79 characters into line from system input file
CONT-CHAR = $?
YES
NO
CALL OUTPUT (COUNT-1) call MOVUP (count-1)
Print the card: Read 1 char into COUNT-CHAR, 79 chars into LINE
DO WHILE CONT-CHAR = '$'
PRINT THE CARD; Read 1char into CONT-CHAR, 79 chars into line.
CHECK CONT-CHAR;
PRINT THE CARD
GO TO AGAIN
CHOP OFF COLS. 73-80
DELETE LEADING AND TRAILING BLANKS
Append LINE to NCARD
RETURN

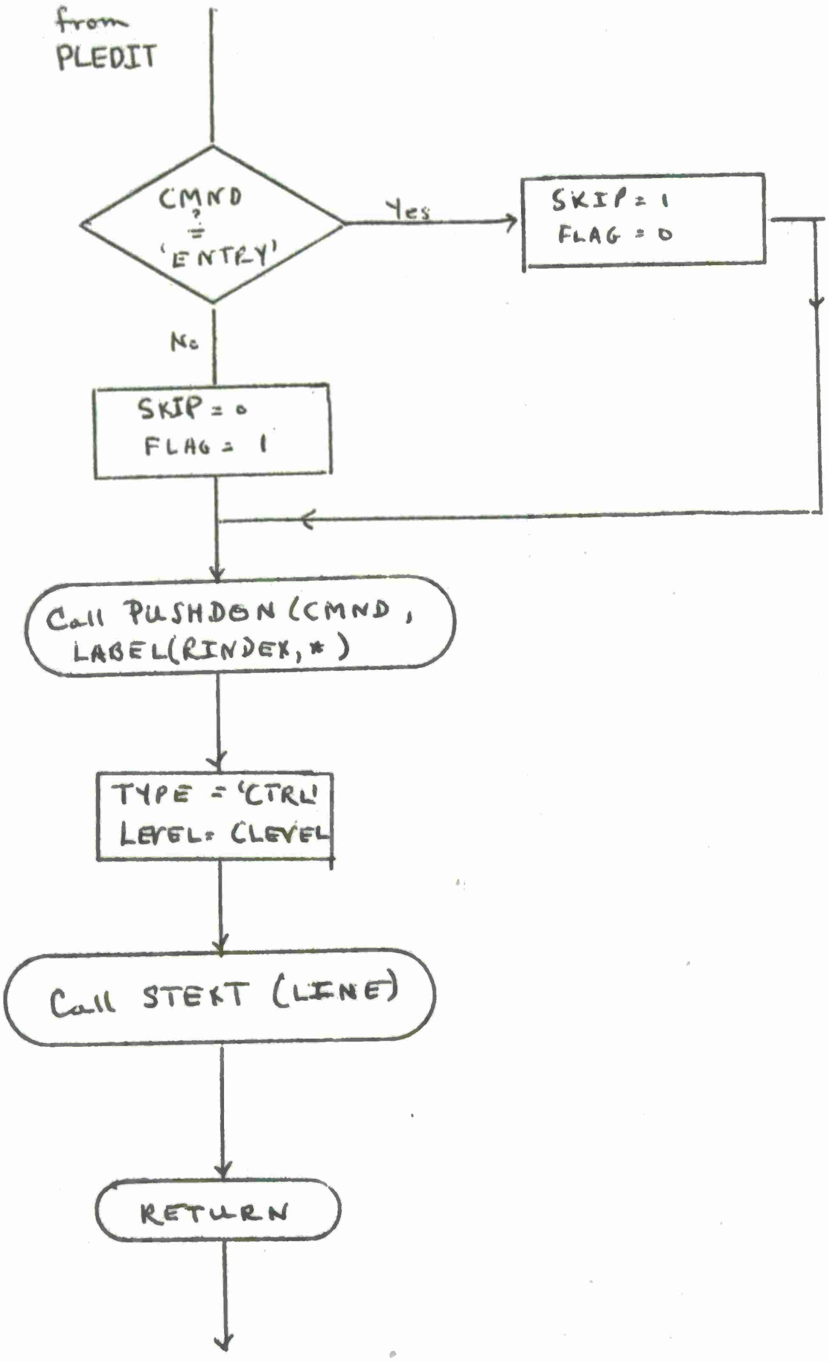
ASSIGN Subroutine



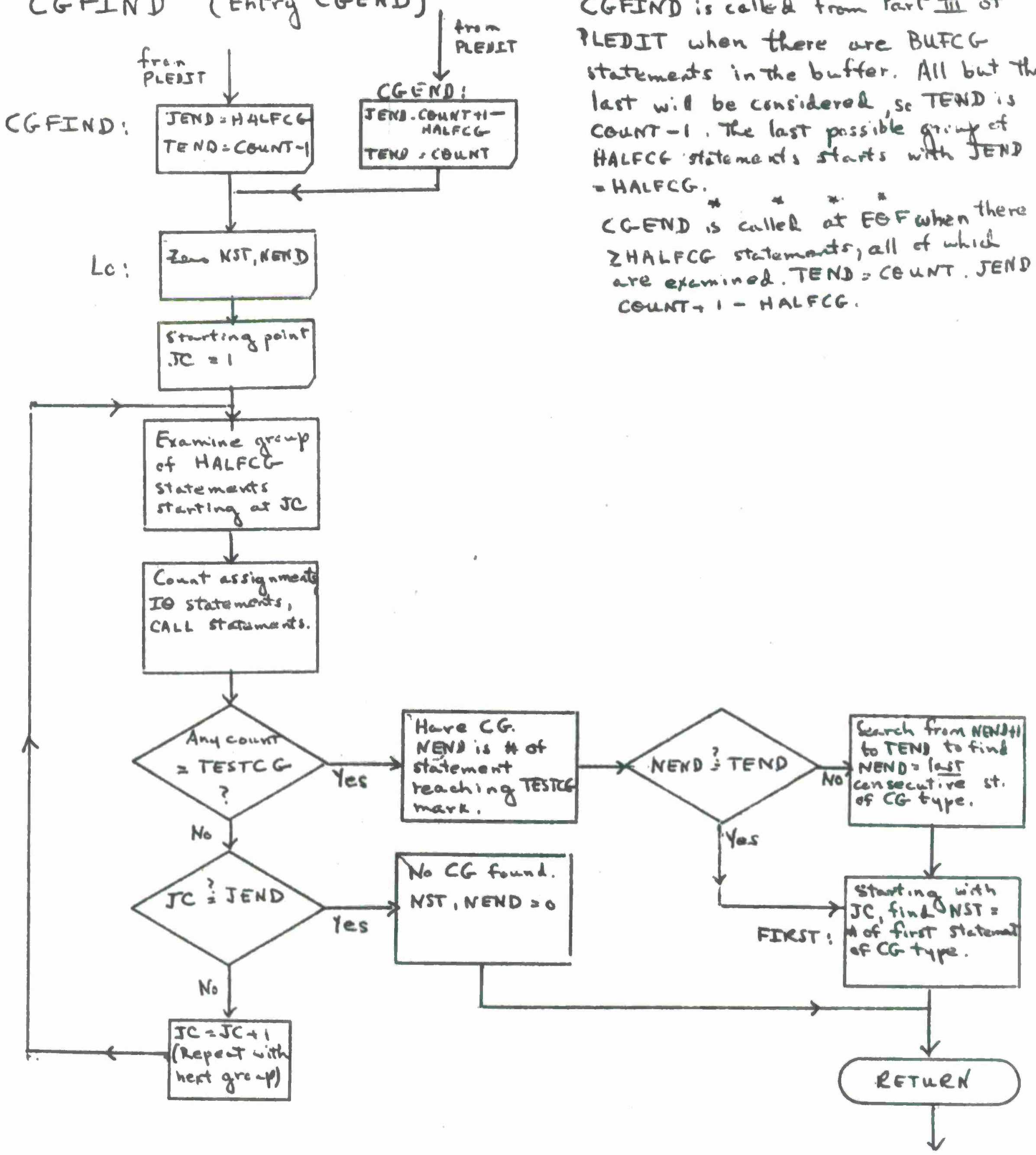
ENTERS 'FREE' statement as a special case.

BLOCK

Called by PLEDIT to process PROCEDURE, BEGIN, DO, ENTRY statements and enter them on pushdown list. Nesting level is not incremented for ENTRY.



CGFIND (Entry CGEND)


from PLEDIT
from PLEDIT
CGFIND:
JEND = HALFCG
TEND = COUNT-1
CGEND:
JEND = COUNT+1 − HALFCG
TEND = COUNT
Lc:
Zero NST, NEND
Starting point JC = 1
Examine group of HALFCG statements starting at JC
Count assignments IO statements, CALL statements.
Any count ≥ TESTCG ?
Yes
Have CG. NEND is # of statement reaching TESTCG mark.
NEND ? TEND
No
Search from NEND+1 to TEND to find NEND = last consecutive st. of CG type.
Yes
FIRST:
Starting with JC, find NST = # of first statement of CG type.
No
JC ? JEND
Yes
No CG found. NST, NEND = 0
No
JC = JC+1 (Repeat with next group)
RETURN

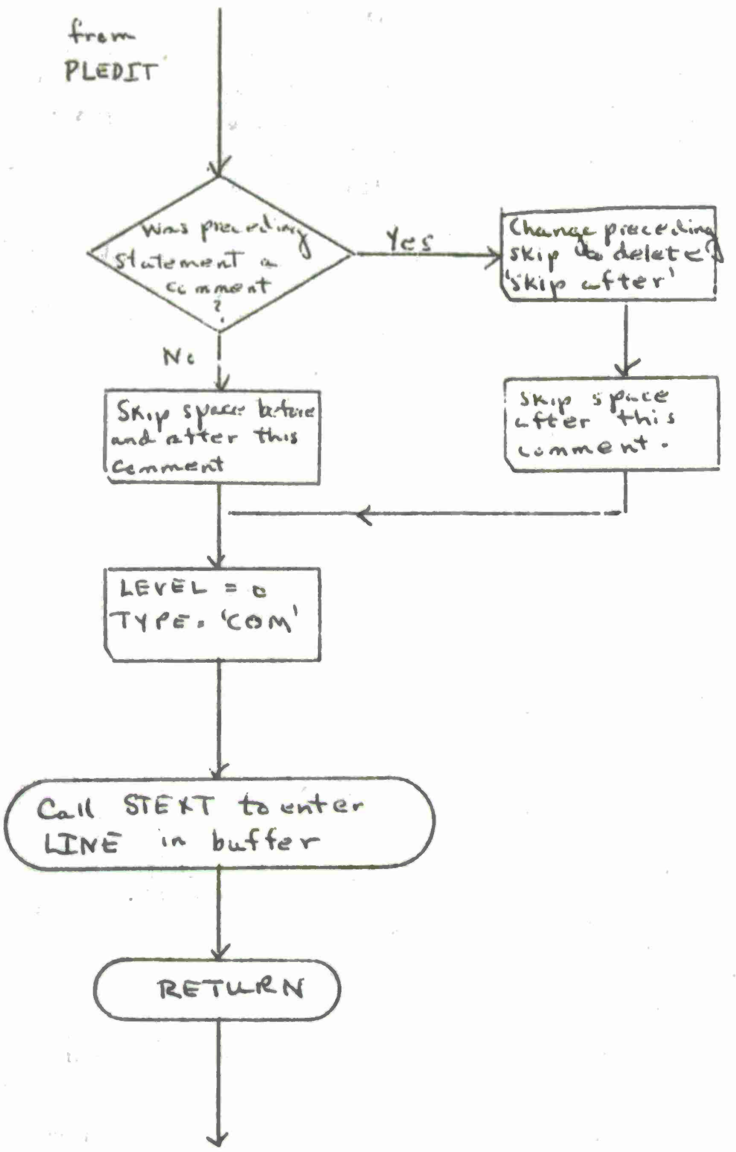
CGFIND is called from Part III of PLEDIT when there are BUFCG statements in the buffer. All but the last will be considered, so TEND is COUNT−1. The last possible group of HALFCG statements starts with JEND = HALFCG.

* * * *

CGEND is called at EOF when there are ≥HALFCG statements, all of which are examined. TEND = COUNT. JEND = COUNT+1 − HALFCG.

COMMENT

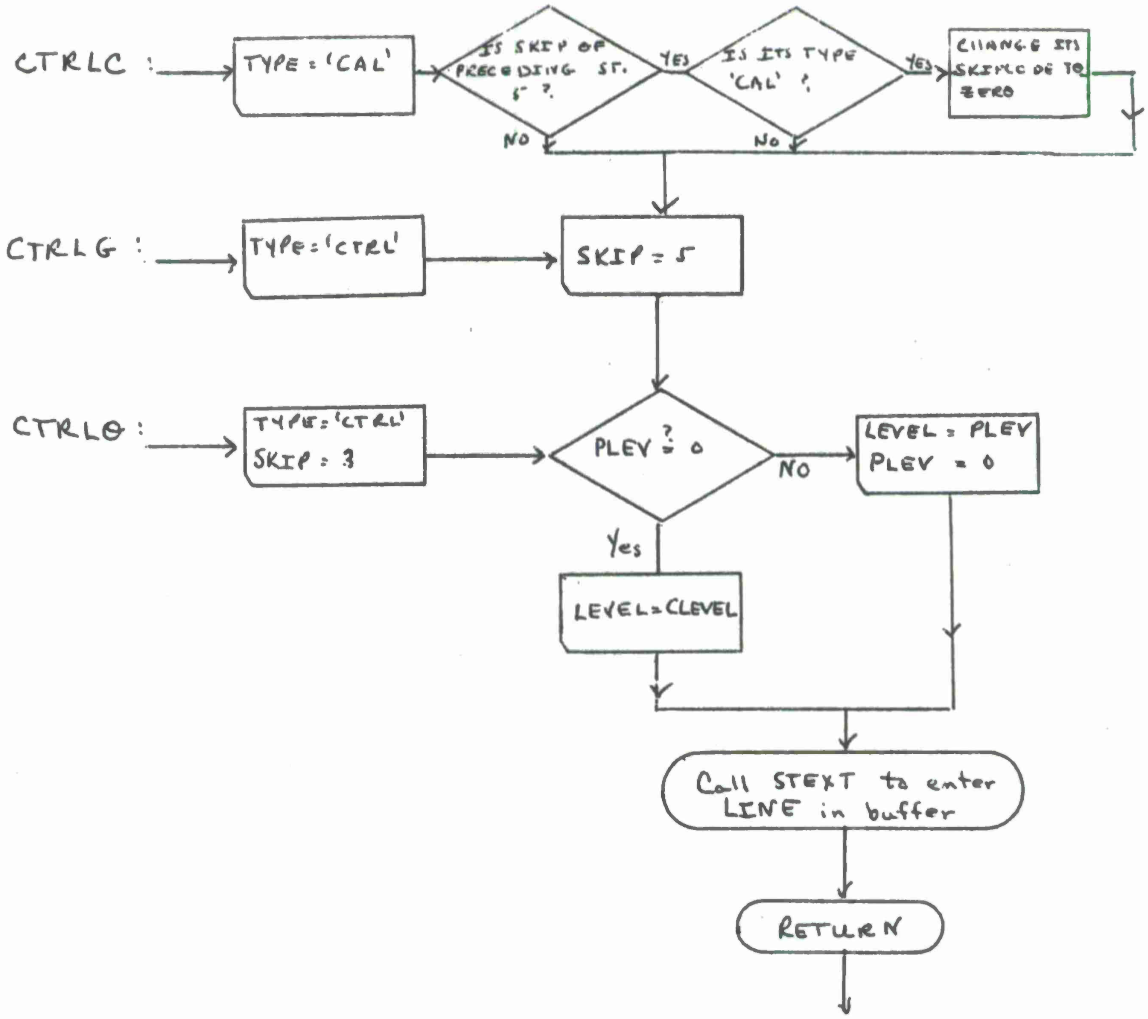
Processes comments. Space before first and after last in a group of comments.


from PLEDIT
Was preceding statement a comment ?
Yes
Change preceding skip to delete 'skip after'
No
Skip space before and after this comment
Skip space after this comment.
LEVEL = 0
TYPE = 'COM'
Call STEXT to enter LINE in buffer
RETURN

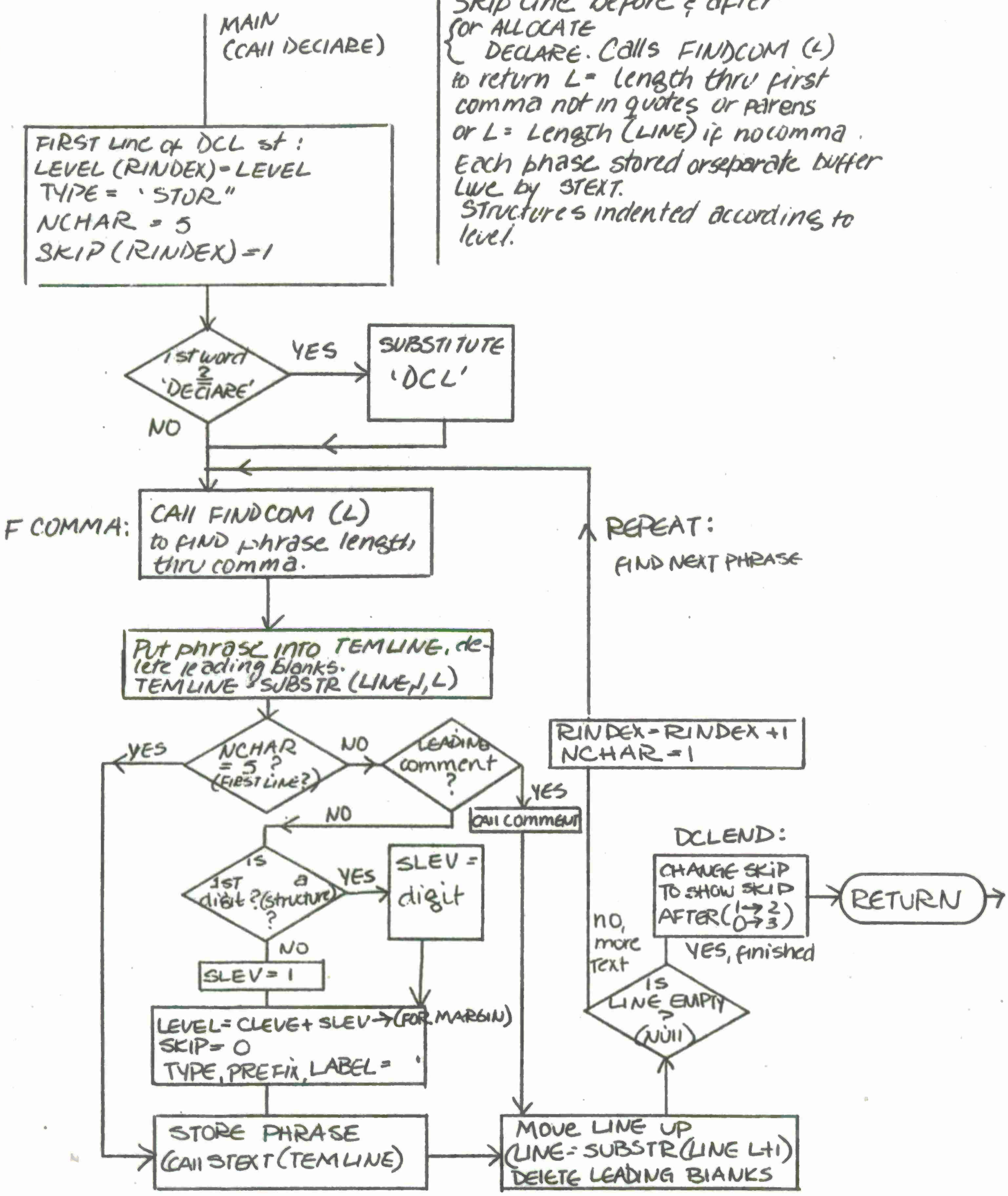
CTRLC (Entries CTRLG, CTRLO)

CTRLC called by PLEDIT for CALLR, CTRLG for GOTO st., CTRLO for other control statements · WAIT, DELAY, STOP, RETURN, EXIT. A CALL (or group of CALLS), GOTO are followed by skip + dotted line. Others by skip only

If statement is part of ELSE st., nesting level is in PLEV, not CLEVEL

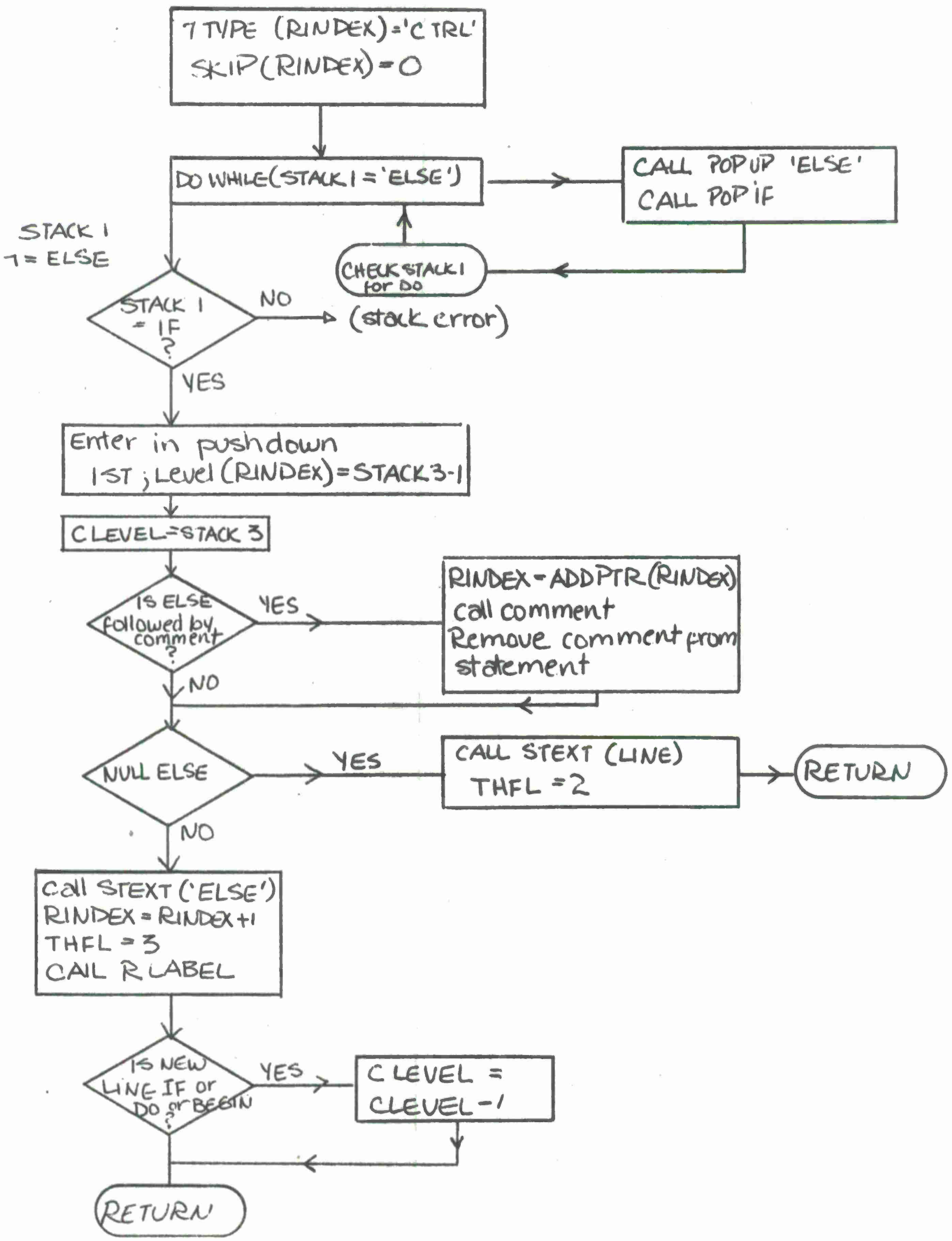


SUB DECLARE



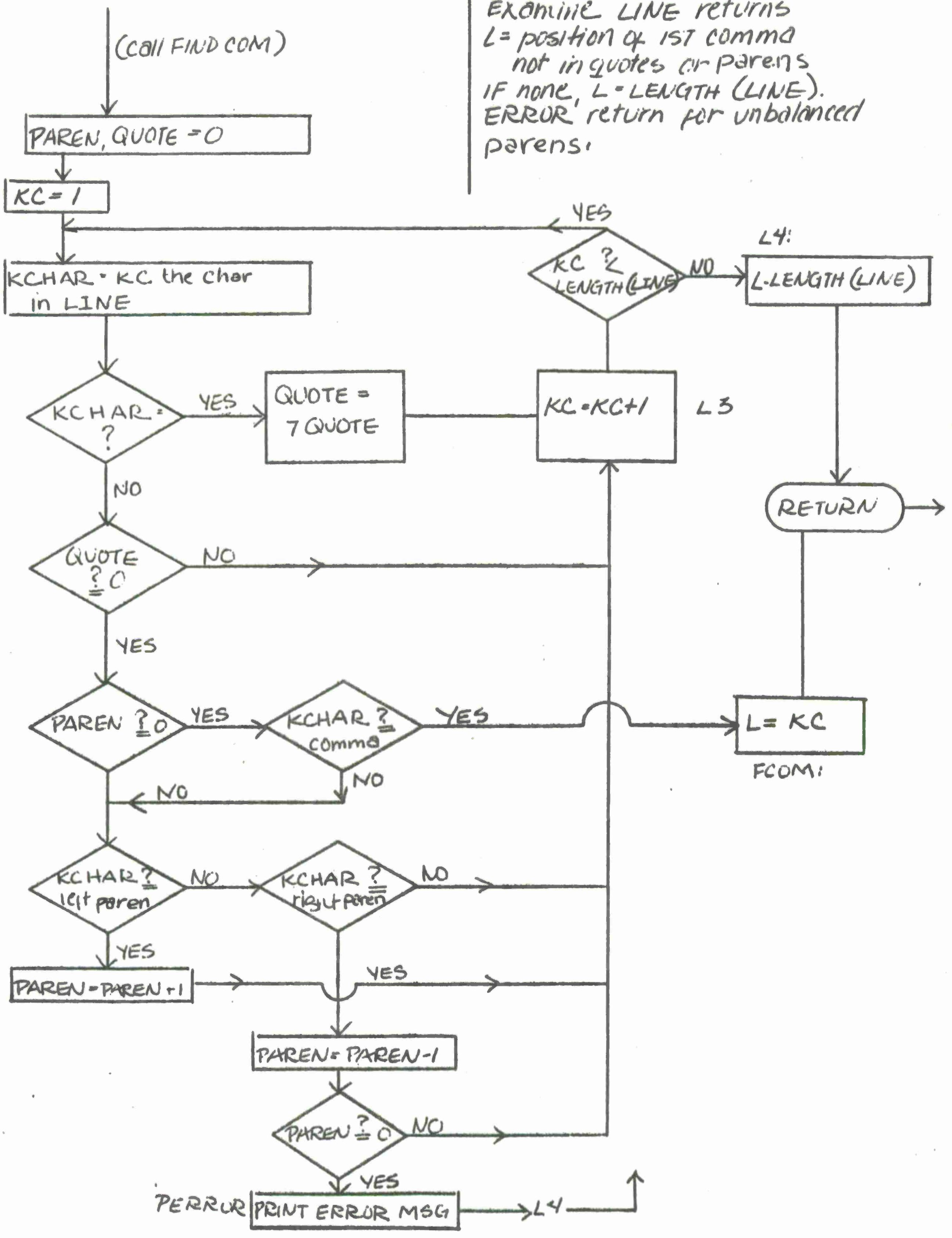
Skip line before & after
{for ALLOCATE
{ DECLARE. Calls FINDCOM (L)
to return L= length thru first
comma not in quotes or parens
or L= Length (LINE) if no comma.
Each phase stored or separate buffer
line by STEXT.
Structures indented according to
level.

ELSE (called by PUSHPULL)

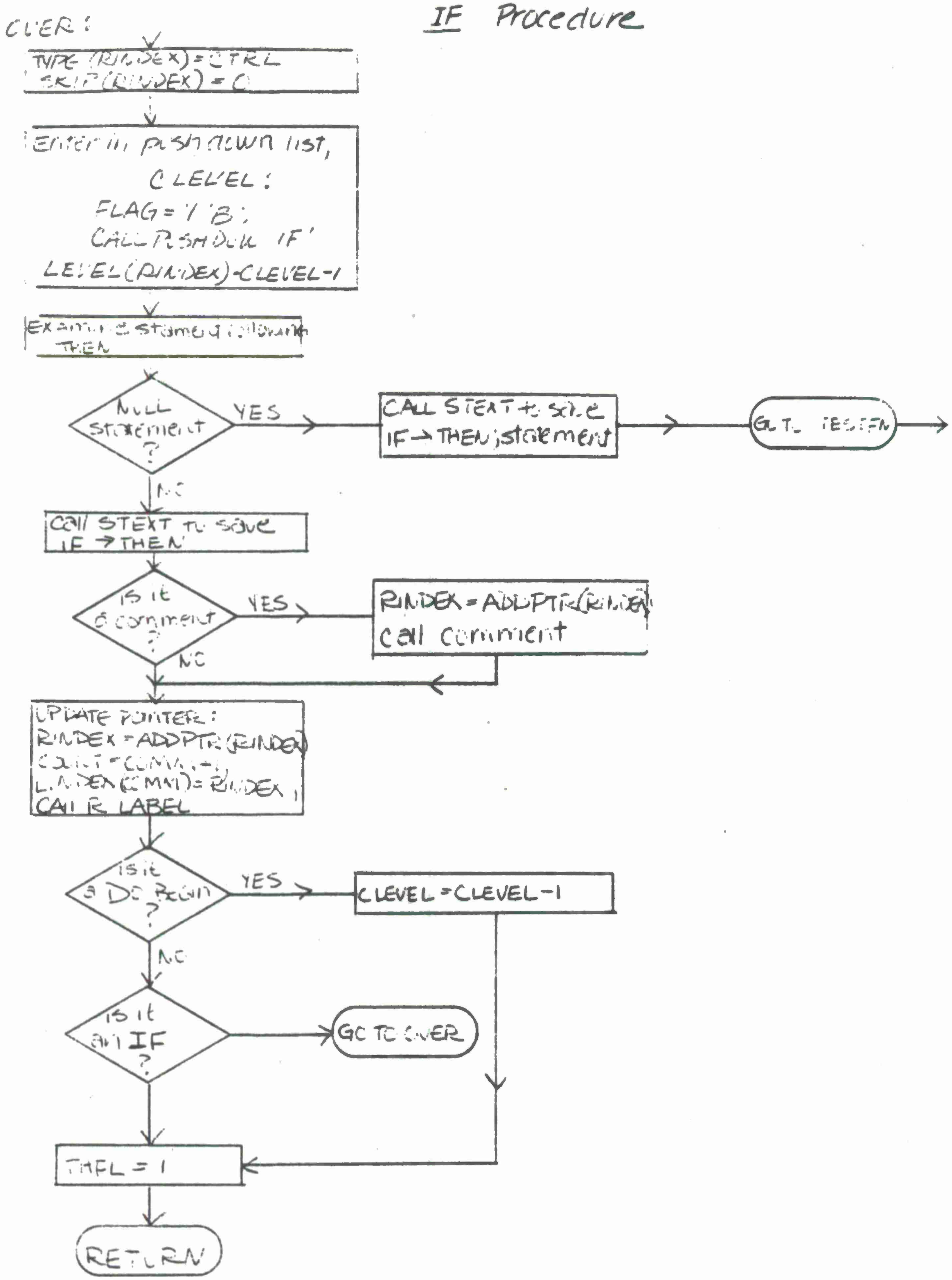


SUB FINDCOM

Examine LINE returns L = position of 1ST comma not in quotes or parens. IF none, L = LENGTH (LINE). ERROR return for unbalanced parens.

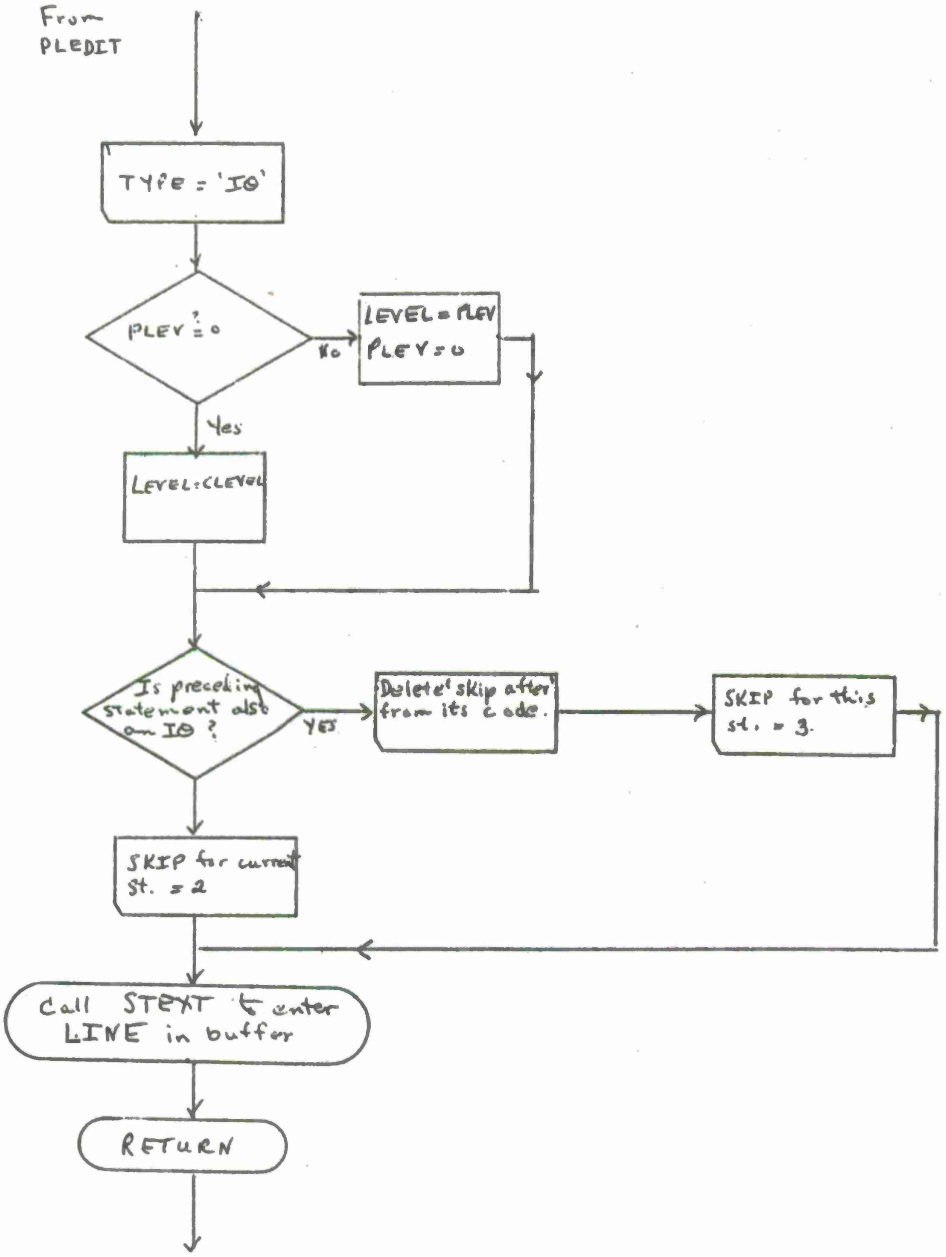


IF Procedure



IOSUB

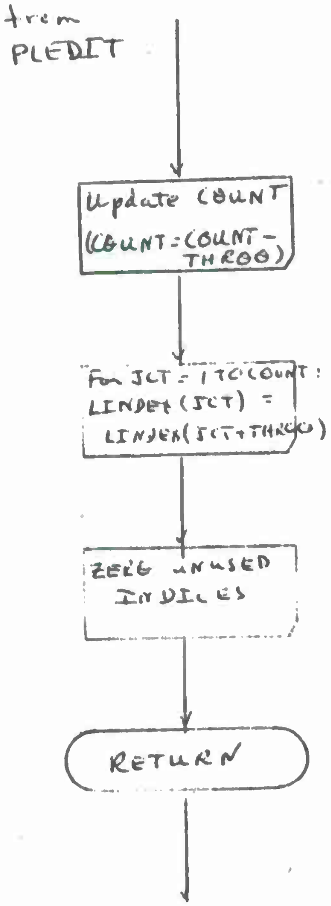
Skip space before + after
each IO or group of IOs.
Level = PLEV (part of ELSE st)
or CLEVEL.



MOVUP(THROO)

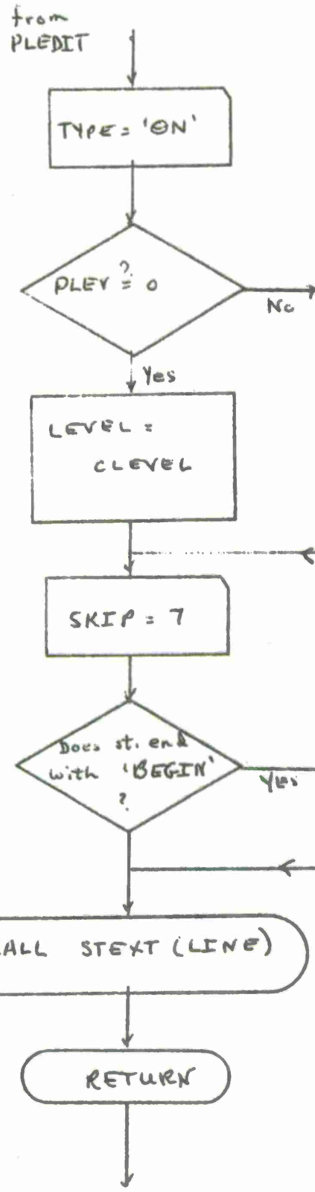
Called by PLEDIT after OUTPUT to 'move up' remaining buffer lines Actually only LINDEX, the table of pointers to first lines of statements, is changed. COUNT = # of statements currently in buffer. THROO = # of last statement to be deleted.

Written for LINDEX dimensioned at 30


from PLEDIT
Update COUNT (COUNT = COUNT - THROO)
For JCT = 1 TO COUNT: LINDEX(JCT) = LINDEX(JCT+THROO)
ZERO UNUSED INDICES
RETURN

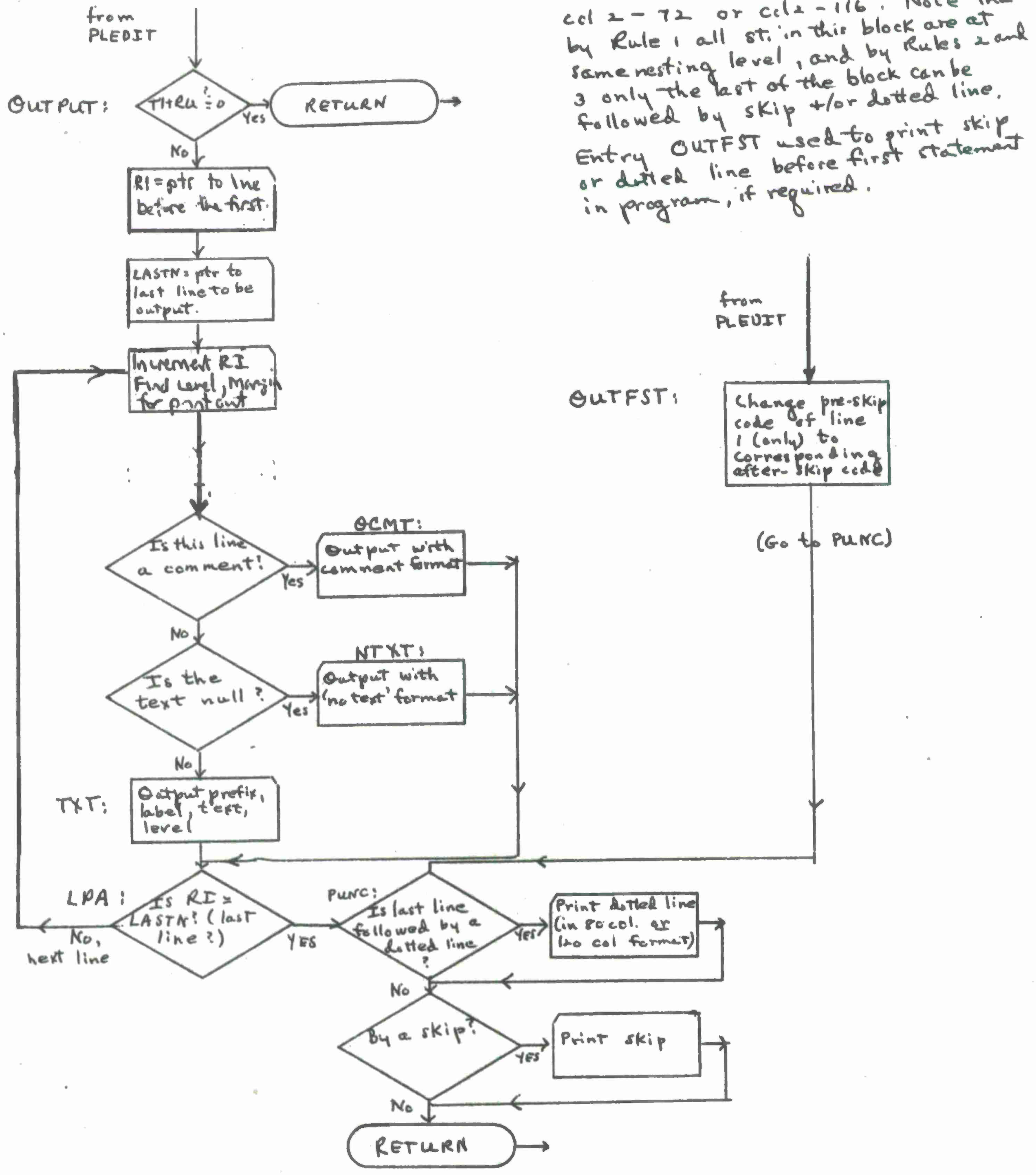
ONSUB

For 'ON CONDITION' statements, SIGNAL and REVERT. Precedes each by blank space and dotted line, follows by dotted line, unless 'ON' st. includes a BEGIN group, when ONFLAG is set to 1, and the terminating dotted line is implemanted after the corresponding END.



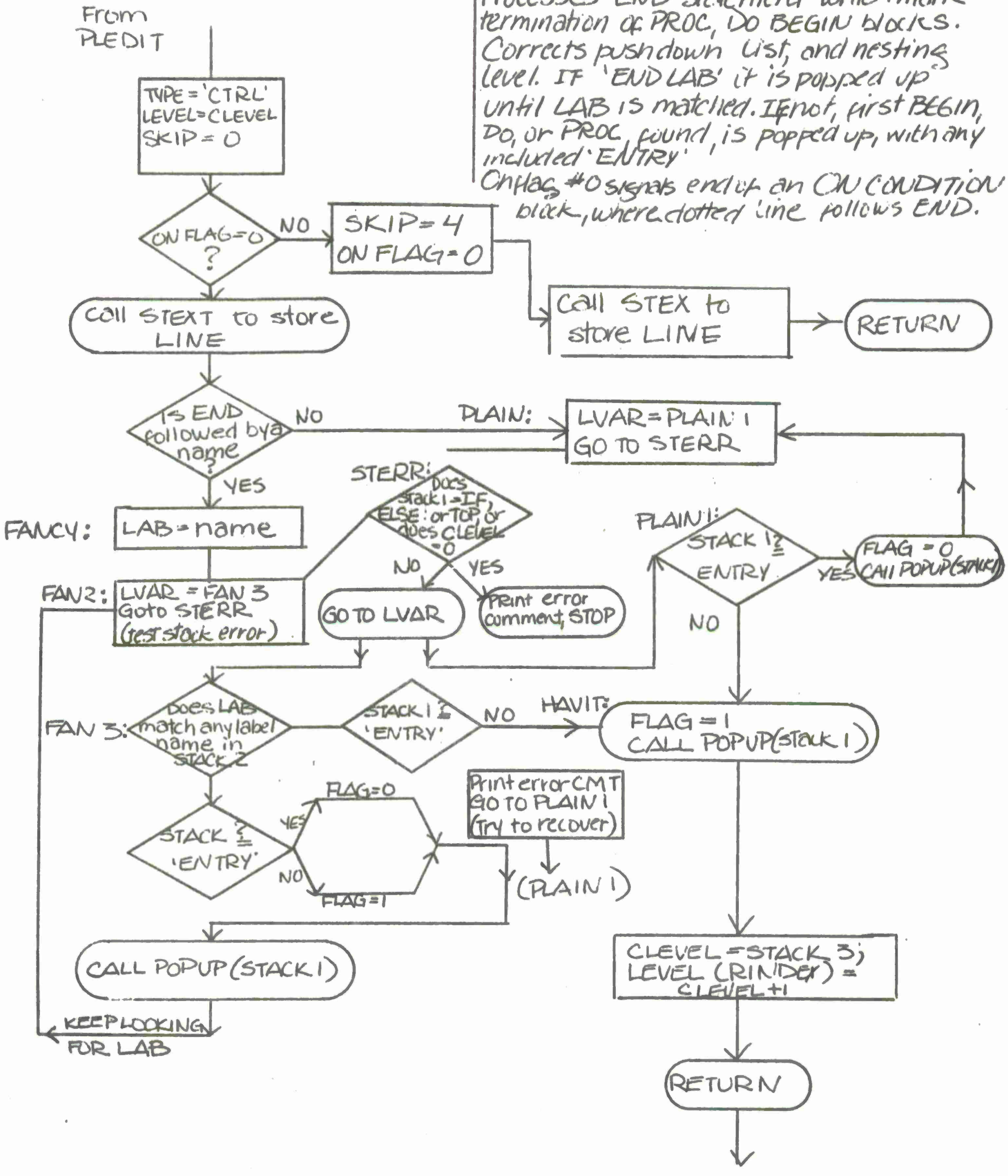
OUTPUT (THRU)
(Entry OUTFST)

Prints (system file) statements 1—THRU from buffer. Written for 80 column or 120 col output, with col 1 blank and all text within col 2 – 72 or col 2 – 116. Note that by Rule 1 all st. in this block are at same nesting level, and by Rules 2 and 3 only the last of the block can be followed by skip +/or dotted line. Entry OUTFST used to print skip or dotted line before first statement in program, if required.

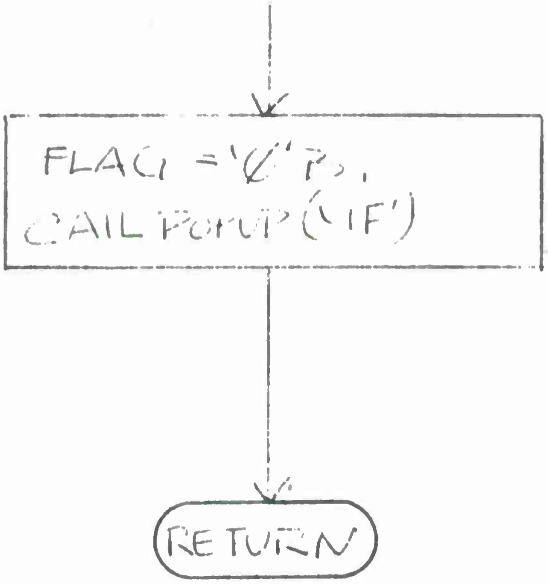

from PLEDIT
OUTPUT:
THRU ?= 0
Yes
RETURN
No
R1 = ptr to line before the first.
LASTN = ptr to last line to be output.
Increment R1 Find Level, Margin for print out
OCMT:
Is this line a comment?
Yes
Output with comment format
No
NTXT:
Is the text null?
Yes
Output with 'no text' format
No
TXT:
Output prefix, label, text, level
LPA:
Is R1 = LASTN? (last line?)
No, next line
YES
PUNC:
Is last line followed by a dotted line?
Yes
Print dotted line (in 80 col. or 120 col format)
No
By a skip?
YES
Print skip
No
RETURN
from PLEDIT
OUTFST:
Change pre-skip code of line 1 (only) to corresponding after-skip code
(Go to PUNC)

PEND

Processes END statement which mark termination of PROC, DO BEGIN blocks. Corrects pushdown list, and nesting level. IF 'END LAB' it is popped up until LAB is matched. If not, first BEGIN, DO, or PROC found, is popped up, with any included 'ENTRY'
Onflag #0 signals end of an ON CONDITION block, where dotted line follows END.

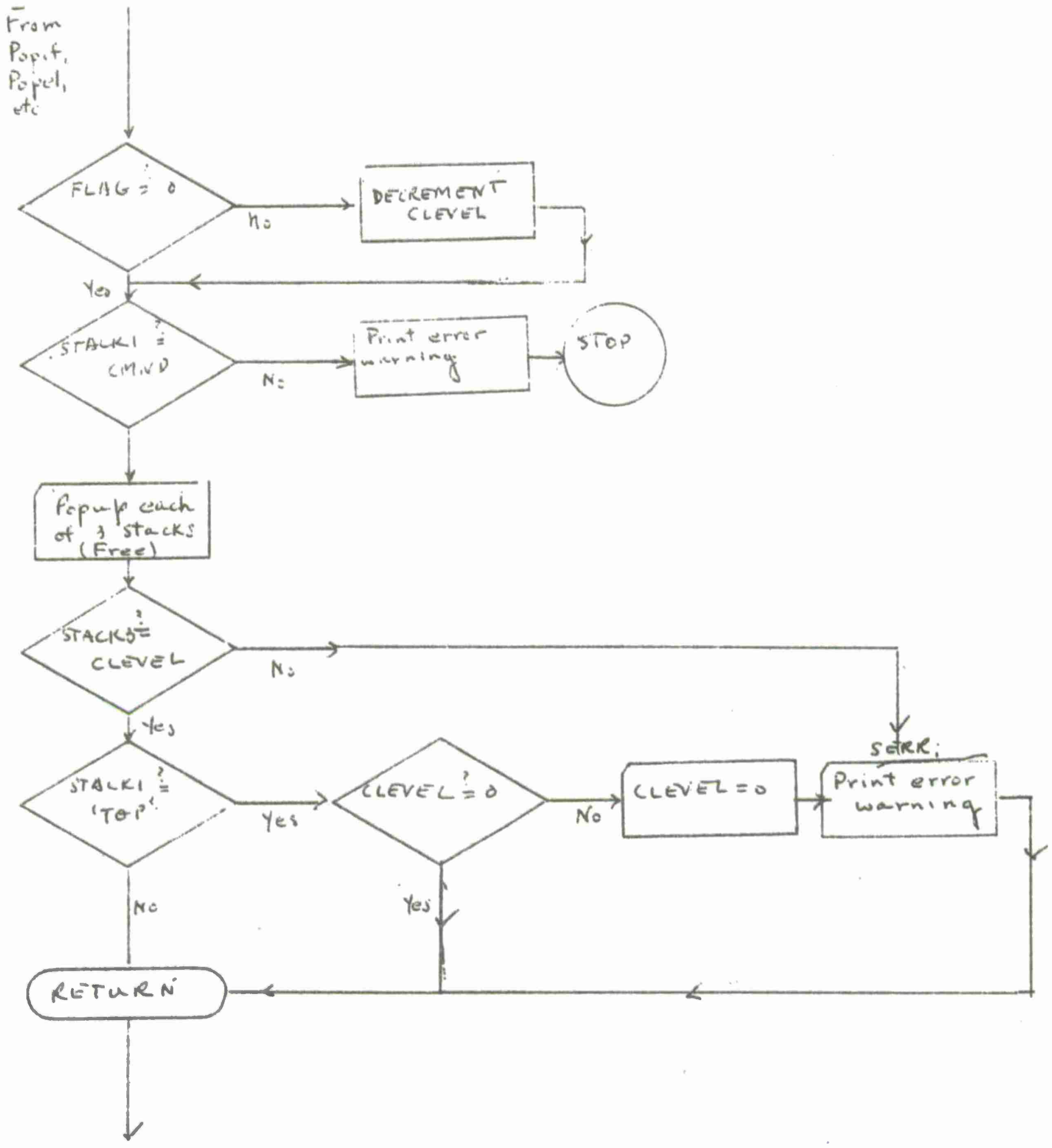


POPIF Subroutine

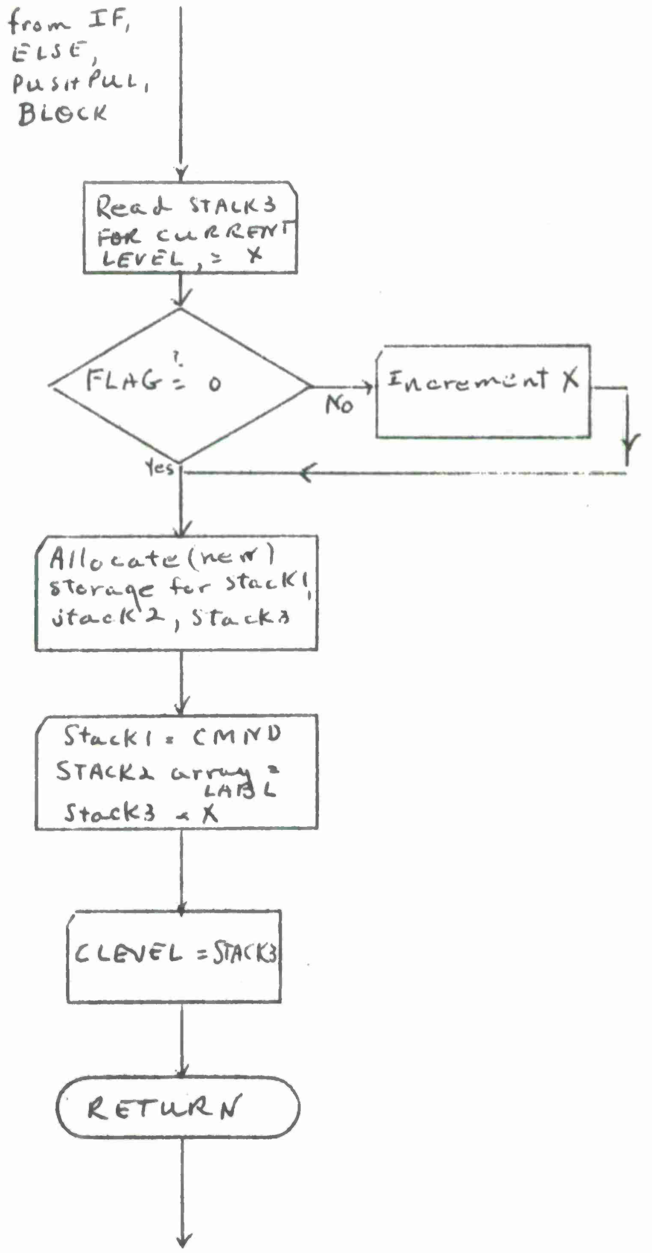


POPUP (CMND)

Pops up 1 level in each of 3 stack lists. Decrements CLEVEL if FLAG = 1,



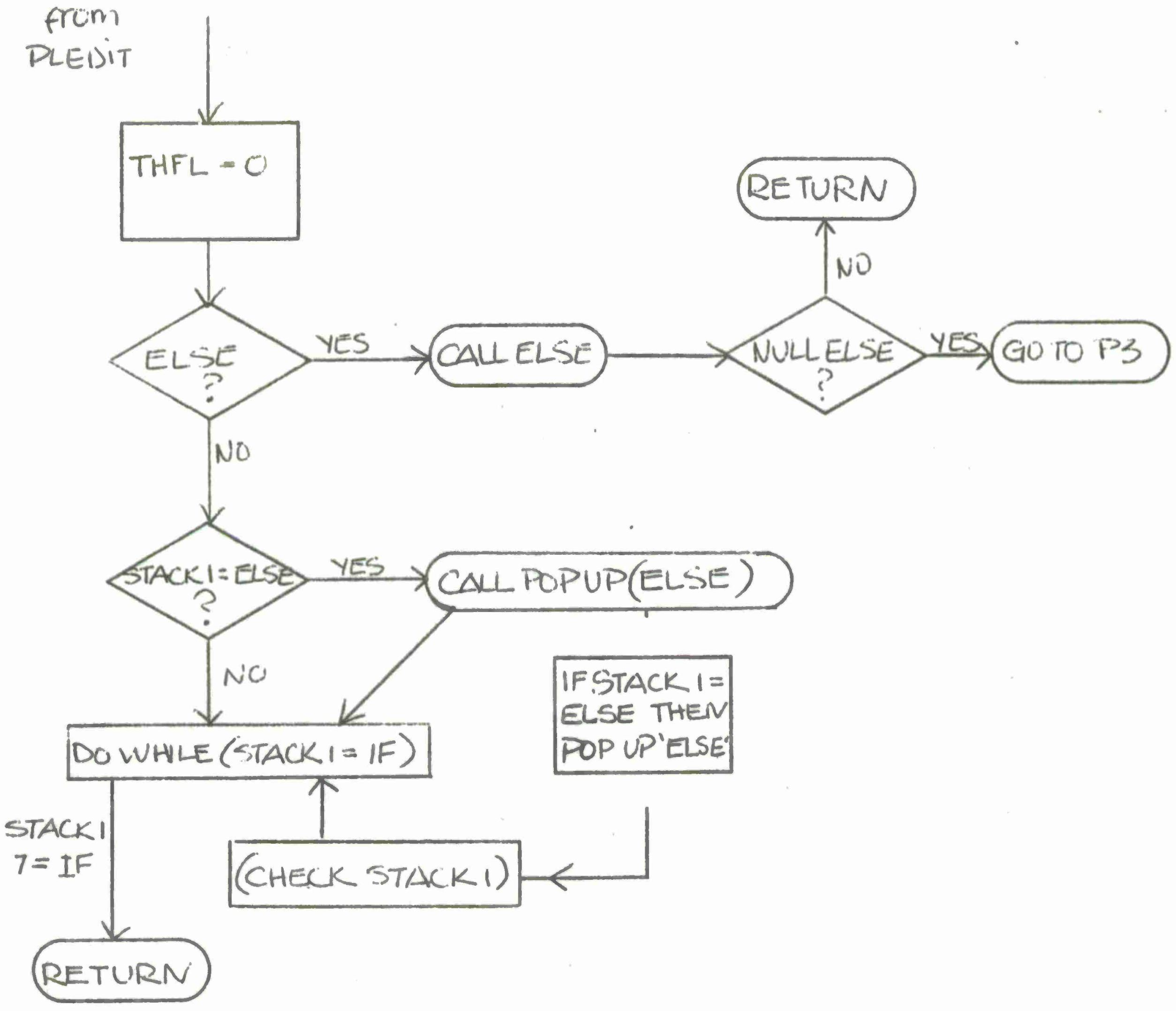
PUSHDON (CMND, LABL);



IF FLAG=1 CLEVEL IS INCREMENTED. PUSHDON ALLOCATES NEW STORAGE FOR STACK1, STACK2, STACK3 (OPERATION, LABEL(S), LEVEL), PUSHING PREVIOUS VALUES DOWN.

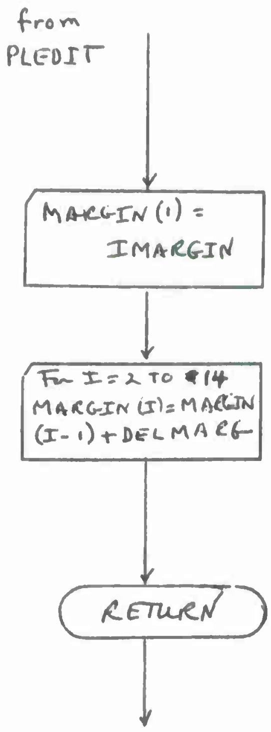
PUSHPUL

Checks on present status of IF... ELSE structures and updates pushdownlists if necessary.



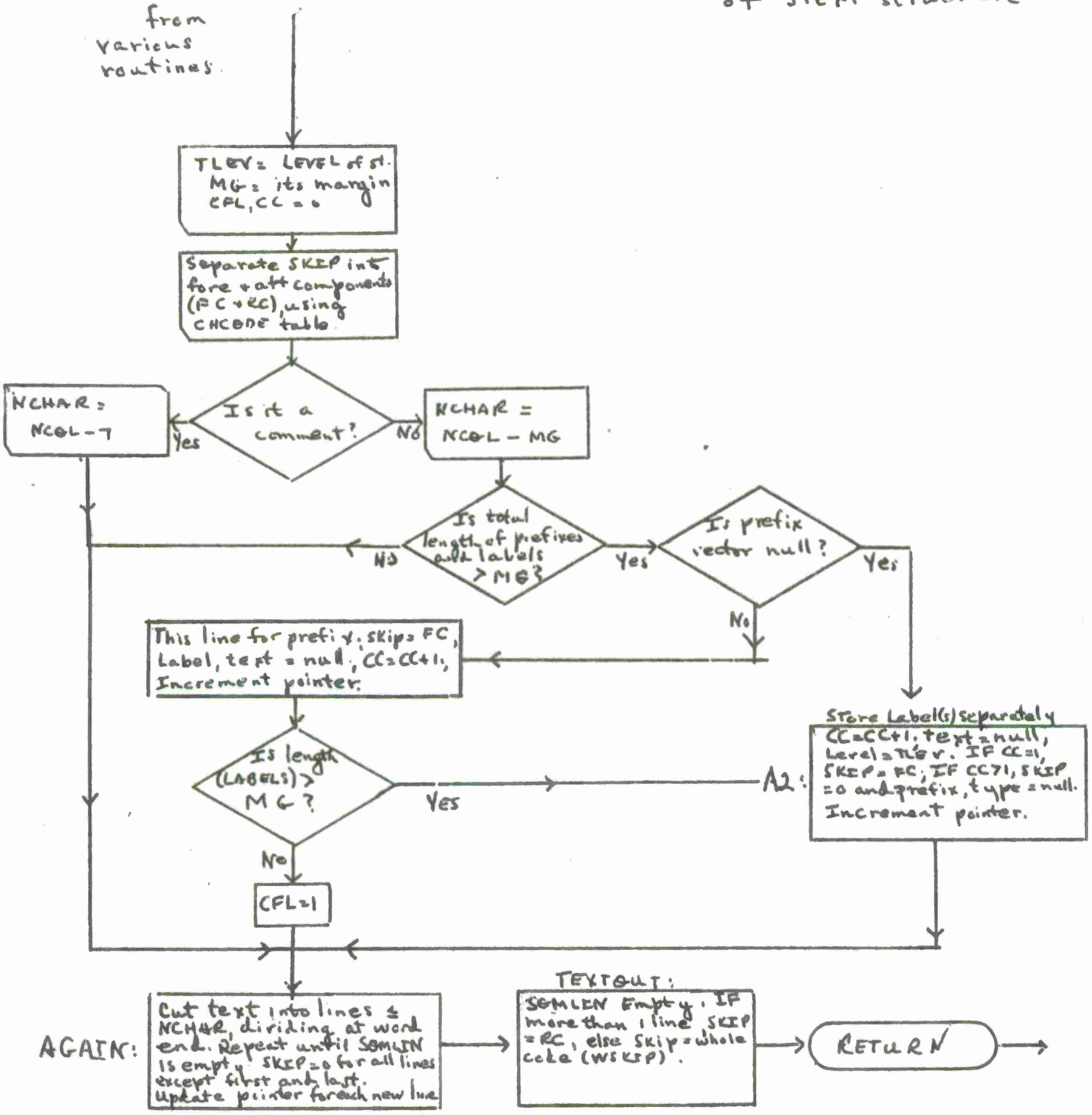
SMARG

Level 0 (comment statements) margin is defined in the declaration st. (PLEDIT) as 1. IMARGIN and DELMARG are read in.



STEXT (SOMLIN)

See Notes for explanation of STEXT structure


from various routines.
TLEV= LEVEL of st.
MG= its margin
CFL,CC = 0
Separate SKIP int fore + aft components (FC + RC), using CHCODE table
Is it a comment?
Yes
No
NCHAR = NCOL-7
NCHAR = NCOL - MG
Is total length of prefixes and labels > MG?
No
Yes
Is prefix vector null?
Yes
No
This line for prefix; skip= FC, Label, text = null, CC=CC+1, Increment pointer.
Is length (LABELS) > MG?
Yes
No
A2:
Store Label(s) separately
CC=CC+1. text=null, Level=TLEV. IF CC=1, SKIP= FC; IF CC>1, SKIP =0 and prefix, type = null. Increment pointer.
CFL=1
AGAIN:
Cut text into lines ≤ NCHAR, dividing at word end. Repeat until SOMLIN is empty. SKIP=0 for all lines except first and last. Update pointer for each new line
TEXTOUT:
SOMLIN Empty. IF more than 1 line SKIP = RC, else Skip = whole code (WSKIP).
RETURN

APPENDIX C

CONCEPTUAL GROUPINGS PROGRAM FOR PL/1 (GP-P)

5. SOURCE PL/1 PROGRAM OF (GP-P) UNGROUPED

```
DESCRIPTION          PL/I FORMATTER
ADDED TO MASTER      C7/19/74
LAST DATE COPIED       NCNE
LAST UPCATED           NCNE

PASSWORD             PWGC
PROGRAMMER           BRYAN
TYPE PARAMETER       COMPCAT
EXEC PARAMETER       PLICPT
CPTION PARAMETER     NCSYM,NOXREF,NOLISTX
JOBNAME              NONE
SLAMP PARAMETER      NO


   PLEDIT: PROCEDURE CPTIONS(MAIN):                                           00000010          09/02/74
   DCL TLINE VAR CONTROLLED:                                                  00000011          09/02/74
   DCL STATEMENT_SIZE FIXED DECIMAL(7):                                       00000012          09/02/74
   DCL MAX_LINES FIXED DECIMAL(7):                                            00000013          09/02/74
   DCL SECNO CHAR(8):                                                         00000014          09/02/74
   DCL CONT_CHAR CHAR(1):                                                     00000015          09/02/74
   DECLARE SYSPRINT PRINT ENV(MEDIUM(SYSLST.1403) F RECSIZE(133)              00000016          09/02/74
    BUFFERS(2)):                                                              00000017          09/02/74
   DECLARE CCIN STREAM INPUT ENV(MEDIUM(SYS004,2540) F RECSIZE(80)):          00000018          09/02/74
               /* PART 1 INITIALIZATION */                                    00000020
               /* BUFFER STORAGE */                                           00000030
               /* LINDEX IS POINTER TABLE TO BUFFER */                        00000040
               /* BUFFER IS 6 ARRAYS, EACH DIMENSIONED 50 */                  00000050
               /* LABEL PREFIX ALLOW MAX OF 5 TO A ST. */                     00000060
               DCL LINDEX(20) FIXED BIN(15,0) INIT((20)0),                    00000070
                   TEXT(*) CHAR(120) VAR CONTROLLED,                          00000080          09/02/74
                   PREFIX(*,*) CHAR(31) VAR CONTROLLED,                       00000081          09/02/74
                   LABEL(*,*) CHAR(31) VAR CONTROLLED,                        00000082          09/02/74
                   TYPE(*) CHAR(4) VAR CONTROLLED,                            00000083          09/02/74
                   LEVEL(*) DEC FIXED(2) CONTROLLED,                          00000084          09/02/74
                   SEQ#(*) CHAR(8) CONTROLLED,                                00000085          09/02/74
                   SKIP(*) BIN FIXED CONTROLLED:                              00000086          09/02/74
               /* COUNT = NUMBER OF CURRENT STATEMENT,                        00000140
                  RINDEX IS POINTER TO CURRENT BUFFER                         00000150
                  LINE      */                                                00000160
               DCL(CCUNT,RINDEX) BIN FIXED(15,0)INIT(0):                      00000170
               /* MARGIN TABLE ANS PARAMETERS FOR PRINTING                    00000180
                  AND COMPUTING MARGINS    */                                 00000190
              DCL MARGIN(0:15) BIN FIXED(15,0)INIT(1),                        00000200          09/02/74
                  (DELMARG INIT(2), IMARGIN INIT(9),                          00000210
             NCOL INIT(80)) BIN FIXED(15,0),                                  00000220
                   PAGENO DEC FIXED(3,0) INIT(1):                             00000230
               /* PUSHDOWN LISTS : STACK1 = OPERATION,                        00000240
                  STACK2 HAS LABEL(S), STACK3 = LEVEL    */                   00000250
               DCL (STACK1 CHAR(5) VAR,                                       00000260
              STACK2(5) CHAR(31) VAR,                                         00000270
                    STACK3 DEC FIXED(2)) CONTROLLED:                          00000280
               /* 80-COL RECORDS ARE READ, 1 AT A TIME, INTO                  00000290
             NCARC. EACH STATEMENT, AS IT IS SEPARATED                 00000300
             FROM NCARD, IS STORED IN LINE FOR ANALYSIS.               00000310
             TEMLINE IS WORKING STRING STORAGE   */                    00000320
DCL NCARD VAR CONTROLLED.                                              00000330                09/02/74
LINE VAR CONTROLLED.                                                   00000331                09/02/74
TEMLINE VAR CONTROLLED:                                                00000332                09/02/74
          /* MISC. VARIABLES : FLAGS, ETC.   */                        00000360
          DCL (COMFLG,THFL) FIXED BIN(15,0) INIT(0),                   00000370
               CMNC CHAR(10) VAR,                                      00000380
              (NV1,NV2,NST,NEND,THISN,LASTN,PT)                        00000390
              BIN FIXED(15,0);                                         00000400
          DCL (FLAG,FINIS,LEVF,ONFLAG,RFLAG)                           00000410
              BIT(1) INIT('0'B);                                       00000420
          /* CLEVEL IS CURRENT LEVEL. PLEV IS                          00000430
              SAVED PRECEDING LEVEL USED IN SOME                       00000440
              ELSE STATEMENTS   */                                     00000450
          DCL (CLEVEL,PLEV) DEC FIXED(2) INIT(0);                      00000460
          /* BINARY CONSTANTS   */                                     00000470
          DCL (ZB INIT(0), ONEB INIT(1),                               00000480
              TWOB INIT(2)) BIN FIXED(15,0);                           00000490
          /* LOCK FCR CG WHEN COUNT = BUFCG(.=20).                     00000500
             TESTCG = NUMBER DEFINING A CG ( .=HALFCG,                 00000510
             WHICH IS BUFCG/2)   */                                    00000520
          DCL (BUFCG INIT(20),HALFCG,                                  00000530
              TESTCG INIT(7)) BIN FIXED(15,0);                         00000540
          /* CHCODE TABLE SEPARATES SKIPCODE INTO FORE                 00000550
             AND AFT COMPONENTS   */                                   00000560
          DCL CHCODE(0:9,2) BIN FIXED(15,0)                            00000570
               INIT( 0,0, 1,0, 1,3, 0,3, 0,4, 0,5,                     00000580
                     6,0, 6,4, 8,0, 8,4);                              00000590
          /* STATEMENT TYPES   */                                      00000600
          DCL LTYPE(7) CHAR(4) VAR INIT                                00000610
              ('AST', 'IO', 'CAL', 'CTRL', 'STOR',                     00000620
               'CN', 'COM');                                           00000630
          /* BREAK CHARATERS AND NULL ARRAY   */                       00000640
          DCL BLANK CHAR(1) INIT(' ');                                 00000650
          DCL BRK(8) CHAR(2) VAR INIT                                  00000660
               ('''', ',', ';', '(', ')', '/*', '*/', ':');            00000670
          DCL MTLAB(5) CHAR(2) VAR INIT((5)'');                        00000680
          /* INPUT IS SYSTEM FILE, CARDS.80COL,                        00000690
              PROVIDE FOR END OF FILE   */                             00000700
          ON ENDFILE (SYSIN)                                           00000710
              BEGIN:                                                   00000720
                FINIS = '1'B;                                          00000730
                GC TO ENDRD;                                           00000740
          END;                                                         00000750
          /* READ IN LINESIZE(80 OR 120) FOR OUTPUT,                   00000760
                MARGIN PARAMETERS   */                                 00000770
ON ENDFILE(CCIN) BEGIN;                                                00000780                09/02/74
 PUT EDIT('NOT ENOUGH CONTROL INFORMATION SUPPLIED (SYS004).')(A);     00000781                09/02/74
PUT EDIT('ITEM 1: LENGTH OF PRINTLINE: SUGGESTED 120')(SKIP(1),A);     00000782                09/02/74
 PUT EDIT('ITEM 2: BEGINNING MARGIN: SUGGESTED 9')(SKIP(1),A);         00000783                09/02/74
 PUT EDIT('ITEM 3: MARGIN STEP SIZE: SUGGEST 5')(SKIP(1),A);           00000784                09/02/74
 PUT EDIT('ITEM 4: THE NUMBER 20')(SKIP(1),A);                         00000785          09/02/74
 PUT EDIT('ITEM 5: THE NUMBER 7')(SKIP(1),A);                          00000786          09/02/74
 PUT EDIT('ITEM 6: MAX # CHARACTERS PER PL/I STATEMENT: SUGGESTED 800')00000787          09/02/74
 (SKIP(1),A);                                                          00000788          09/02/74
 PUT EDIT('ITEM 7: MAX # EDITED LINES PER PL/I STATEMENT: SUGGESTED '||00000789          09/02/74
 '50')(SKIP(1),A);                                                     00000790          09/02/74
 PUT EDIT('PROGRAM TERMINATED.')(SKIP(1),A);                           00000791          09/02/74
 SIGNAL ERROR;                                                         00000792          09/02/74
END;                                                                   00000793          09/02/74
OPEN FILE(CCIN),FILE(SYSIN);                                           00000794          09/02/74
 GET FILE(CCIN) LIST(NCOL,IMARGIN,DELMARG);                            00000795          09/02/74
       /* DEFINE OPTIONS FOR OUTPUT (SYSPRINT)    */                   00000796
      OPEN FILE(SYSPRINT);                                             00000800          09/02/74
       /* ACTION AT PAGE END    */                                     00000820
       ON ENDPAGE (SYSPRINT)                                           00000830
            BEGIN;                                                     00000840
            PAGENO = PAGENO+1;                                         00000850
            PUT FILE (SYSPRINT) PAGE EDIT                              00000860
              ('PAGE ', PAGENO)(COL(NCOL-8),A,F(3));                   00000870
            END;                                                       00000880
       /* TITLE FIRST PAGE    */                                       00000890
          PUT FILE(SYSPRINT) EDIT('SOURCE EDITED BY PLEDIT',           00000900          09/02/74
              'PAGE ',PAGENO)                                          00000901          09/02/74
            (SKIP(2),COL(10),A,COL(NCOL-8),A,F(3));                    00000920
       PUT SKIP(2);                                                    00000930
       /* INITIALIZE PUSHDOWN STACK    */                              00000940
       ALLOCATE STACK1 INIT('TOP'),                                    00000950
              STACK2(5) INIT((5)''),                                   00000960
              STACK3 INIT(0);                                          00000970
       /* ADJUST LINELENGTH FOR LEVEL PRINTOUT    */                   00000980
       IF NCOL = 80 THEN                                               00000990
            NCCL = 72;                                                 00001000
       ELSE NCOL = 116;                                                00001010
       /* PARAMETERS FOR CG SEARCH    */                               00001020
      GET FILE(CCIN) LIST(BUFCG,TESTCG);                               00001030          09/02/74
       HALFCG = BUFCG/TWOB;                                            00001040
       IF TESTCG > HALFCG THEN                                         00001050
            DO;                                                        00001060
              PUT SKIP LIST('TESCG MUST BE < HALF BUFCG');             00001070          09/02/74
            STOP;                                                      00001090
          END;                                                         00001100
/* GET STATEMENT SIZE  */                                              00001101          09/02/74
      GET FILE(CCIN) LIST(STATEMENT_SIZE,MAX_LINES);                   00001102          09/02/74
       ALLOCATE TEXT(MAX_LINES),PREFIX(MAX_LINES,5),                   00001103          09/02/74
          LABEL(MAX_LINES,5), TYPE(MAX_LINES),                         00001104          09/02/74
          SEQ#(MAX_LINES),                                             00001105          09/02/74
          LEVEL(MAX_LINES), SKIP(MAX_LINES);                           00001106          09/02/74
       TEXT='';                                                        00001107          09/02/74
       PREFIX='';                                                      00001108          09/02/74
       LABEL='';                                                       00001109          09/02/74
       TYPE='';                                                        00001110          09/02/74
       LEVEL=0;                                                        00001111          09/02/74
       SKIP=0;                                                         00001112          09/02/74
```

```
  ALLOCATE TLINE CHAR(STATEMENT_SIZE) INIT('');                    00001113        09/02/74
  ALLOCATE NCARD CHAR(STATEMENT_SIZE) INIT('');                    00001114        09/02/74
  ALLOCATE LINE CHAR(STATEMENT_SIZE) INIT('');                     00001115        09/02/74
  ALLOCATE TEMLINE CHAR(STATEMENT_SIZE) INIT('');                  00001116        09/02/74
   /* CREATE MARGIN TABLE   */                                     00001117
   CALL SMARG;                                                     00001120
   /* PART II     */                                               00001130
   /* HERE WE START EDITING, 1 STATEMENT AT A TIME    */           00001140
   /* UPDATE POINTERS AND INDEX    */                              00001150
GO:  COUNT = COUNT+ONEB;                                           00001160
   RINDEX = ADDPTR(RINDEX);                                        00001170
   LINDEX(COUNT) = RINDEX;                                         00001180
   /* GET 1ST. STORE CONDITION PREFIX AND/OR LABEL    */           00001190
   /* IF EOF, EMPTY BUFFER AND END    */                           00001200
       CALL READ;                                                  00001210
   /* COME HERE ON EOF    */                                       00001220
ENDRD:     IF (FINIS) THEN                                         00001230
      IF (¬RFLAG) THEN                                             00001240
       DO;                                                         00001250
        COUNT = COUNT-ONEB;                                        00001260
        RINDEX = SUBPTR(RINDEX);                                   00001270
        GO TO IFCG;                                                00001280
      END;                                                         00001290
     ELSE                                                          00001300
      DO;                                                          00001310
       PUT SKIP LIST('**MISSING CARD(S)**');                       00001320
       GO TO ENDPROG;                                              00001330
      END;                                                         00001340
   /* NOT EOF. ANALYZE STATEMENT FOR TYPE,                         00001350
     GO TO TYPE ROUTINES, UPDATE                                   00001360
     PUSHDOWN LIST, ASSIGN LEVEL AND                               00001370
     SKIPCODE, STORE IN BUFFER    */                               00001380
   /* IS IT COMMENT ?    */                                        00001390
   IF CCMFLG = ONEB THEN                                           00001400
     DO;                                                           00001410
      CALL COMMENT;                                                00001420
      GO TO P3;                                                    00001430
     END;                                                          00001440
   /* NO. UPDATE STACKS    */                                      00001450
   CALL PUSHPUL;                                                   00001460
   /* NULL ST. ?    */                                             00001470
   IF LENGTH(LINE) = ONEB THEN                                     00001480
      GO TO C8;                                                    00001490
   /* THFL IS CLUE TO TYPE    */                                   00001500
   /* COMPLETED 'ELSE' ?    */                                     00001510
   IF THFL = TWOB THEN                                             00001520
    GO TO P3;                                                      00001530        09/02/74
   /* FOR THFL = 0 OR 3    */                                      00001570
    /* IS IT PREPROCESSOR STATEMENT */                             00001571        09/02/74
     IF SUBSTR(LINE,1,1)='%' THEN DO;                              00001572        09/02/74
       COUNT=COUNT-1;                                              00001573        09/02/74
       GO TO TESTFN;                                               00001574        09/02/74
     END;                                                          00001575        09/02/74
```

```
      /* IS IT AN 'IF' ?    */
C1:   CMND = SUBSTR(LINE,1,3);
      IF (CMND = 'IF ')|(CMND = 'IF(')|
          (CMND = 'IF''') THEN
           CALL IF;
      /* IS IT DO, PROC, BEGIN OR ENTRY ?   */
C2:   CMND = SUBSTR(LINE,1,6);
      IF(CMND = 'ENTRY ')|(CMND = 'BEGIN ')|
          (CMND = 'ENTRY:')|(CMND = 'BEGIN:') THEN
            GO TO YESBL;
      CMND = SUBSTR(LINE,1,3);
      IF (CMND = 'DO:')|(CMND = 'DO ') THEN
            GO TO YESBL;
      CMND = SUBSTR(LINE,1,5);
      IF (CMND = 'PROC:')|(CMND = 'PROC ')|
          (CMND = 'PROC(') THEN
            GO TO YESBL;
      CMND = SUBSTR(LINE,1,10);
      IF (CMND ¬= 'PROCEDURE ') & (CMND ¬= 'PROCEDURE:')
           & (CMND ¬= 'PROCEDURE(') THEN
            GO TO CC2;
      /* IT IS A BLOCK COMMAND    */
YESBL:CALL BLOCK(CMND);
      GO TO P3;
      /* TRY FOR 'END'    */
CC2:  CMND = SUBSTR(LINE,1,4);
      IF(CMND = 'END ')|(CMND = 'END:') THEN
          DO;
            CALL PEND;
            GO TO P3;
          END;
      /* IS IT A CONTROL STATEMENT ?   */
C4:   CMND = SUBSTR(LINE,1,5);
      IF(CMND = 'CALL ') THEN
          DO;
            CALL CTRLC;
            GO TO P3;
          END;
      IF(CMND = 'GOTO ') THEN
            GO TO C41;
      IF(CMND = 'WAIT ')|(CMND = 'WAIT;')|
          (CMND = 'STOP ')|(CMND = 'STOP;')|
          (CMND = 'EXIT ')|(CMND = 'EXIT;') THEN
            GO TO C44;
      CMND = SUBSTR(LINE,1,6);
      IF(CMND = 'DELAY ')|(CMND = 'DELAY;') THEN
            GO TO C44;
      IF (CMND = 'GO TO ') THEN
C41:    DO;
            CALL CTRLG;
            GO TO P3;
        END;
      CMND = SUBSTR(LINE,1,7);
      IF (CMND = 'RETURN ')|(CMND = 'RETURN:') THEN
C44:    DO;
            CALL CTRLO;
            GO TO P3;
        END;
      /* NOT CONTROL ST    */
/* IS IT IO ?    */
C5:   CMND = SUBSTR(LINE,1,4);
      IF(CMND = 'GET ')|(CMND = 'PUT ') THEN
            GO TO C55;
      CMND = SUBSTR(LINE,1,5);
      IF(CMND = 'OPEN ')|(CMND = 'READ ') THEN
            GO TO C55;
      CMND = SUBSTR(LINE,1,6);
      IF(CMND = 'CLOSE ')|(CMND = 'WRITE ') THEN
            GO TO C55;
      CMND = SUBSTR(LINE,1,7);
      IF(CMND = 'DELETE ')|(CMND = 'LOCATE ')|
          (CMND = 'FORMAT ')|(CMND = 'FORMAT(')
           THEN
            GO TO C55;
      CMND = SUBSTR(LINE,1,8);
      IF(CMND = 'REWRITE ')|(CMND = 'DISPLAY ')
           THEN
C55:    DO;
            CALL IOSUB;
            GO TO P3;
        END;
      /* IS IT DECLARATION ?    */
C6:   IF(SUBSTR(LINE,1,4) = 'DCL ')|
          (SUBSTR(LINE,1,8) = 'DEFAULT ')|
          (SUBSTR(LINE,1,8) = 'DECLARE ')|
          (SUBSTR(LINE,1,9) = 'ALLOCATE ') THEN
        DO;
            CALL DECLARE;
            GO TO P3;
        END;
      IF(SUBSTR(LINE,1,5) = 'FREE ') THEN
            GO TO C8;
      /* IS IT ON, SIGNAL, REVERT ?    */
C7:   IF(SUBSTR(LINE,1,3) = 'ON ')|
         (SUBSTR(LINE,1,7) = 'SIGNAL ')|
         (SUBSTR(LINE,1,7) = 'REVERT ') THEN
        DO;
            CALL ONSUB;
            GO TO P3;
        END;
      /* NONE OF PRECEDING, CLASS IT ASSIGNMENT    */
C8:   CALL ASSIGN;
      /* PART III    */
      /* RETURNED FROM TYPE SUBS, NOW CHECK
          FOR OUTPUT, GO, ETC. TETURN TO 'GO'
          IF INPUT REMAINS, OR TO END PROGRAM
          IF FINISHED.   */
P3:   THFL = ZB;
      /* POINTER TO FIRST BUFFER LINE OF
          CURRENT STATEMENT   */
      THISN = LINDEX(COUNT);
      NV1 = SKIP(THISN);
      /* IF THIS IS VERY FIRST ST, DO ANY
          PRE-SKIP   */
      IF(COUNT > 1) THEN
            GO TO MORN1;
      NV2 = CHCODE(NV1,ONEB);
      IF NV2 = ZB THEN
            GO TO TESTFN;
      IF(NV2 = ONEB)|(NV2=6)|(NV2 = 8) THEN
            CALL OUTFST(NV2);
      /* SUBTRACT PRE-SKIP FROM CODE   */
        SKIP(THISN) = CHCODE(NV1,TWOB);
            GO TO TESTFN;
      /* NOT LINE 1: PTR TO LAST LINE FO PRECEDING ST   */
MORN1:LASTN = SUBPTR(THISN);
      NV2 = SKIP(LASTN);
      /* RULE1 : FLAG OUTPUT IF CHANGE IN LEVEL   */
RULE1:IF(LEVEL(THISN) = LEVEL(LASTN)) THEN
            LEVF = '0'B;
      ELSE
            LEVF = '1'B;
      /* RULE2 : EXAMINE CURRENT ST. FOR PRESKIP.
          IF FOUND, ADD TO PREV. ST. SUBTRACT HERE   */
RULE2:IF(NV1 = ONEB)|(NV1 = TWOB) THEN
         IF(NV2 = 4)|(NV2 = 5) THEN
           SKIP(LASTN) = 5;
         ELSE
           SKIP(LASTN) = 3;
      ELSE
         IF(NV1 = 6)|(NV1 = 7) THEN
           SKIP(LASTN) = 5;
         ELSE
         IF(NV1 = 8)|(NV1 = 9) THEN
           IF(NV2 = ZB) THEN
            SKIP(LASTN) = 4;
         ELSE
            IF(NV2 = 3) THEN
               SKIP(LASTN) = 5;
      SKIP(THISN) = CHCODE(SKIP(THISN),TWOB);
      /* RULE3 : IMPLEMENT SKIPS FOR PRECEDING ST   */
      /* ALSO OUTPUT IF LEVEL CHANGE
          (RULE1)   */
RULE3:NV2 = SKIP(LASTN);
      IF(LEVF)|(NV2 ¬= ZB) THEN
         DO;
            CALL OUTPUT(COUNT-ONEB);
            CALL MOVUP(COUNT-ONEB);
            GO TO TESTFN;
```

```
                END;
          /* NO OUTPUT YET. LOOK FOR CGIF
              BUFFER = BUFCG OR IF (EOF AND
              BUFFER > HALFCG).   */
    IFCG: IF (COUNT = BUFCG) | ((COUNT >= HALFCG)
               & FINIS) THEN
          GO TO RULE4;
          /* NOT FULL ENOUGH. READ MORE UNLESS
                     NO MORE INPUT   */
    TESTFN:IF(¬FINIS) THEN
                GO TO GO;
          /* EMPTY BUFFER AT END    */
    OUTEND:   CALL OUTPUT(COUNT);
                GO TO ENDPROG;
          /* SEARCH FOR CG    */
    RULE4:IF(FINIS) THEN
                CALL CGEND(NST,NEND);
             ELSE
                CALL CGFIND(NST,NEND);
          /* NOT FOUND, OUTPUT HALF BUFFER
               (OR ALL, IF END OF DATA)    */
          IF(NST = ZB) THEN
               IF(FINIS) THEN
                GO TO OUTEND;
             ELSE
             DO;
             CALL OUTPUT(HALFCG);
             CALL MOVUP(HALFCG);
             GO TO GO;
             END;
          /* FOUNDCG, OUTPUT PRE-CG LINES    */
          IF(NST = ONEB) THEN
                GO TO CGOUT;
          PT = SUBPTR(LINDEX(NST));
          SKIP(PT) = 3;
          NST = NST-ONEB;
          CALL OUTPUT(NST);
          CALL MOVUP(NST);
           /* OUTPUT CG    */
            NEND = NEND-NST;
    CGOUT:  IF(NEND = COUNT) THEN
                GO TO OUTGP;
            PT = SUBPTR(LINDEX(NEND+ONEB));
            SKIP(PT) = 3;
    OUTGP:  CALL OUTPUT(NEND);
            CALL MOVUP(NEND);
                GO TO TESTFN;
          /* ALL INPUT PROCESSED, ALL OUTPUT
              DONE, TELL IT.    */
ENDPROG: PUT PAGE LIST ('**PLEDIT FINISHED**');
/* ******** PAGE 20 ON HANDWRITTEN SHEETS ******** */
```

217

```
      /* SUBROUTINE  READ GETS INPUT AND                         00003710
          BEGINS TO PROCESS STATEMENT  */                        00003720
READ: PROC;                                                      00003730
      /*  GETSTAT PUTS STATEMENT INTO LINE   */                  00003740
      CALL  GETSTAT;                                             00003750
      /* COMMENTS ARE NOT PARSED */                              00003760
      IF (COMFLG = ONEB) THEN                                    00003770
             RETURN;                                             00003780
      /* RLABEL SEPARATES LABEL(S) AND                           00003790
            CONDITION PREFIX(ES)  */                             00003800
      CALL RLABEL;                                               00003810
      /* NOW RETURN IS MADE TO PLEDIT   */                       00003820
     END READ;                                                   00003830
GETSTAT:      PROC;                                              00003840
              DCL CLABL(0:2) LABEL INIT(ISITC,COMT,NOCOMT),      00003850          09/02/74
                        (NCUOT,NCOM) BIT(1) INIT('0'B),          00003860
                        KCH CHAR(1),                             00003870
                        KCC CHAR(2),                             00003880
                        KS BIN FIXED(15,0);                      00003890
              CCMFLG = ZB;                                       00003900
      /* IF MORE TEXT NEEDED, READ NEW RECORD    */              00003910
              IF (NCARD = '') THEN                               00003920
G1:             DO;                                              00003930
                  RFLAG = '0'B;                                  00003940
                  CALL IN;                                       00003950
                  END G1;                                        00003960
      /* SUBROUTINE IN READS SYSIN (SYSTEM FILE)   */            00003970
IN:               PROC;                                          00003980
      /* READ CARD COL2 - 72, APPEND TO NCARD   */               00003990
      /* ON EOF, FINIS = 1 AND RETURN MADE TO ENDRD (IN PLEDIT)*/ 00004000
     AGIN:                                                       00004010          09/02/74
          GET FILE(SYSIN) EDIT(CONT_CHAR,LINE)                   00004011          09/02/74
            (A(1),A(79));                                        00004012          09/02/74
    IF NCARD='' THEN SECNO=SUBSTR(LINE,72);                      00004013          09/02/74
                  LINE = SUBSTR(LINE,1,71);                      00004030
    IF CONT_CHAR='$' THEN DO;                                    00004031          09/02/74
       CALL OUTPUT(COUNT-ONEB);                                  00004032          09/02/74
        CALL MOVUP(COUNT-ONEB);                                  00004033          09/02/74
        PUT EDIT(LINE)(SKIP(1),X(1),A);                          00004034          09/02/74
        GET FILE(SYSIN) EDIT(CONT_CHAR,LINE)                     00004035          09/02/74
          (A(1),A(79));                                          00004036          09/02/74
        LINE=SUBSTR(LINE,1,71);                                  00004037          09/02/74
        DO WHILE(CONT_CHAR¬='$');                                00004038          09/02/74
          PUT EDIT(LINE)(SKIP(1),X(1),A);                        00004039          09/02/74
          GET FILE(SYSIN) EDIT(CONT_CHAR,LINE)                   00004040          09/02/74
            (A(1),A(79));                                        00004041          09/02/74
          LINE=SUBSTR(LINE,1,71);                                00004042          09/02/74
        END;                                                     00004043          09/02/74
        PUT EDIT(LINE)(SKIP(1),X(1),A);                          00004044          09/02/74
         GO TO AGIN;                                             00004045          09/02/74
       END;                                                      00004046          09/02/74
          IF LINE=' ' THEN GO TO AGIN;                           00004047          09/02/74
    DO I=71 TO 1 BY -1 WHILE(SUBSTR(LINE,I,1)=' ');              00004048          09/02/74
    END;                                                            00004049           09/02/74
    IF I<71 THEN I=I+1;                                             00004050           09/02/74
     LINE=SUBSTR(LINE,1,I);                                         00004051           09/02/74
    DO I=1 TO 71 WHILE(SUBSTR(LINE,I,1)=' ');                       00004052           09/02/74
    END;                                                            00004053           09/02/74
     IF I>1 THEN I=I-1;                                             00004054           09/02/74
    LINE=SUBSTR(LINE,I);                                            00004055           09/02/74
    I=LENGTH(NCARD)+LENGTH(LINE);                                   00004056           09/02/74
    IF I>STATEMENT_SIZE THEN DO;                                    00004057           09/02/74
      PUT EDIT('STATEMENT SIZE EXCEEDED.   ')(PAGE,A);              00004058           09/02/74
    PUT EDIT(' NCARD: ',NCARD)(SKIP(1),2 A);                        00004059           09/02/74
    PUT EDIT('  LINE: ',LINE)(SKIP(1),2 A);                         00004060           09/02/74
    STOP;                                                           00004061           09/02/74
     END;                                                           00004062           09/02/74
                  NCARD = NCARD || LINE;                            00004063
                  END IN;                                           00004064
      /* BRANCH FOR TEST(COMFLG = 0), COMMENT(1), NO(2)    */       00004065
C1:   GO TO CLABL(COMFLG);                                          00004070
      /* DELETE LEADING BLANKS    */                                00004080
      /* IS IT COMMENT ?    */                                      00004090
ISITC:    NCARD = SUBSTR(NCARD,VERIFY(NCARD,BLANK));                00004100
            IF INDEX(NCARD,BRK(6)) = 1 THEN                         00004110
              DO;                                                   00004120
              CCMFLG = ONEB;                                        00004130
              GO TO COMT;                                           00004140
              END;                                                  00004150
      /* STATEMENT IS NOT CCMMENT, WILL END IN                      00004160
                  SEMICOLON (NOT IN QUOTES OR                       00004170
                  COMMENT)    */                                    00004180
            CCMFLG = TWOB;                                          00004190
NOCOMT:     DO KS = 1 TO LENGTH(NCARD);                             00004200
            KCH=SUBSTR(NCARD,KS,1);                                 00004210
                  IF(KCH = BRK(3)) & (¬NQUOT) & (¬NCOM)             00004220
                    THEN                                            00004230
                    GO TO FBRK;                                     00004240
                  IF (KCH = BRK(1)) THEN                            00004250
                    NQUOT = ¬NQUOT;                                 00004260
          ELSE IF KS¬=LENGTH(NCARD) THEN                            00004270           09/02/74
IF (SUBSTR(NCARD,KS,2)=BRK(6) | SUBSTR(NCARD,KS,2)=BRK(7)) THEN     00004271           09/02/74
                      NCOM = ¬NCOM;                                 00004300
            END NOCOMT;                                             00004310
      /* NO ENDBREAK FOUND, GET MORE TEXT    */                     00004320
MORE:       RFLAG = '1'B;                                           00004330
            CALL IN;                                                00004340
            GO TO C1;                                               00004350
      /* FOUND ENDBREAK, STORE ST. IN LINE    */                    00004360
FBRK: LINE = SUBSTR(NCARD,1,KS);                                    00004370
      /* CLEAR ST. FORM NCARD    */                                 00004380
      NCARD = SUBSTR(NCARD,KS+1);                                   00004390
      RETURN;                                                       00004400
      /* COMMENT ST., FIND END    */                                00004410
COMT: DO KS = 3 TO LENGTH(NCARD) - 1;                               00004420
            KCC = SUBSTR(NCARD,KS,2);                               00004430
            IF (KCC = BRK(7)) THEN
              DO;
              KS = KS + 1;
              GO TO FBRK;
            END COMT;
      /* NO END BREAK FOUND   */
            GO TO MORE;
      END GETSTAT;
      /* RLABEL IS CALLED BY PLEDIT, IF AND
           ELSE TO SEPARATE ALL PREFIXES   */
RLABEL:     PROC;
      DCL (KLA,NLAB INIT (0)) BIN FIXED(15,0),
            KH    CHAR;
      /* UP TO 31 CHAR IN A PREFIX   */
      /* FIND ANY CONDITION PREFIX(ES)   */
      CALL SPREFX;
SPREFX:     PROC;
            DCL (IP,NP INIT(0)) BIN FIXED (15,0);
      /* SET PREFIX ARRAY TO NULL   */
                  PREFIX(RINDEX,*) = MTLAB;
ISITP:      NP = NP+ONEB;
                  LINE = SUBSTR(LINE,VERIFY(LINE,BLANK));
             IF SUBSTR(LINE,1,1)¬=BRK(4) THEN
              RETURN;
      /* FOUND PREFIX START, LOOK FOR END   */
LPP:              DO IP = 2 TO LENGTH(LINE) - 1;
                    IF SUBSTR(LINE,IP,2)  ¬=');' THEN
            GO TO DEND;
            PREFIX(RINDEX,NP) = SUBSTR(LINE,1,IP + 1);
            LINE = SUBSTR(LINE,IP + 2);
      /* REPEAT FOR ANOTHER   */
            IF NP < 5 THEN
              GO TO ISITP;
            ELSE RETURN;
DEND:    END LPP;
      /* NO END, ERROR.  TRY TO GO ON   */
          PUT SKIP LIST('**UNBALANCED PARENS IN PREFIX**');          09/02/74
            IP = INDEX(LINE,BRK(8));
            IF IP = 0 THEN
              STOP;
            PREFIX(RINDEX,NP) = SUBSTR(LINE,1,IP);
            LINE = SUBSTR(LINE,IP + 1);
            IF NP < 5 THEN
                  GO TO ISITP;
      END SPREFX;
      /* LOOK FOR LABEL(S).  COLON MUST COME
          BEFORE BLANK, QUOTE OR LEFT PAREN   */
      LABEL(RINDEX,*) = MTLAB;
      /* ALWAYS DELETE LEADING BLANKS   */
RL1:  LINE = SUBSTR(LINE,VERIFY(LINE,BLANK));
      NLAB = NLAB + 1;
RLP:  DO KLA = 1 TO LENGTH(LINE) - 1;
            KH = SUBSTR(LINE,KLA,1);
```

220

```
                IF (KH = BRK(1)) | (KH = BLANK) | (KH = BRK(4))                       00004980
                THEN                                                                   00004990
                      RETURN;                                                          00005000
                IF KH = BRK(8) THEN                                                    00005010
                  DO;                                                                  00005020
                LABEL(RINDEX,NLAB) =                                                   00005030
                      SUBSTR(LINE,1,KLA);                                              00005040
                LINE = SUBSTR(LINE,KLA+1);                                             00005050
                IF NLAB < 5 THEN                                                       00005060
                   GO TO RL1;                                                          00005070
                END;                                                                   00005080
          END RLP;                                                                     00005090
          END RLABEL;                                                                  00005100
          /* FUNCTICN ADDPTR AND SUBPTR ARE CALLED                                     00005110
              TO FIND POINTER TO NEXT (PRECEDING) LINE                                 00005120
              IN BUFFER. BUFFER SIZE IS SET TO 50    */                                00005130
    ADDPTR:     PROC (PT) RETURNS (BIN FIXED(15,0));                                   00005140
          /* INCREMENT BUFFER POINTER    */                                            00005150
          DCL(PNEXT,PT) BIN FIXED(15,0);                                               00005160
          PNEXT = PT + ONEB;                                                           00005170
        IF PNEXT > MAX_LINES THEN                                                      00005180                 09/02/74
                PNEXT = ONEB;                                                          00005190
          RETURN (PNEXT);                                                              00005260
          END ADDPTR;                                                                  00005270
    SUBPTR:     PROC(PT) RETURNS (BIN FIXED(15,0));                                    00005280
          /* FINDS POINTER TO PRECEDING BUFFER LINE    */                              00005290
          DCL  PT BIN FIXED(15,0);                                                     00005300
          IF (PT - ONEB)> ZB THEN                                                      00005310
                RETURN (PT - ONEB);                                                    00005320
          ELSE                                                                         00005330
              RETURN(MAX_LINES);                                                       00005340                 09/02/74
          END SUBPTR;                                                                  00005350
          /* ASSIGNMENT STATEMENTS, FREE ST., ETC.    */                               00005360
    ASSIGN:     PROC;                                                                  00005370
          /* ARRIVES HERE BY FALLING THROUGH ALL OTHER                                 00005380
              CLASSIFICATION TESTS OR AS 'FREE' ST.    */                              00005390
          IF(LENGTH(LINE)<6) | (SUBSTR(LINE,1,5)  ¬= 'FREE ')                          00005400
                THEN                                                                   00005410
                TYPE(RINDEX) = LTYPE(1);                                               00005420
          ELSE                                                                         00005430
                TYPE(RINDEX) = LTYPE(5);                                               00005440
          SKIP(RINDEX) = 0;                                                            00005450
          LEVEL(RINDEX) = CLEVEL;                                                      00005460
          CALL STEXT(LINE);                                                            00005470
          END ASSIGN;                                                                  00005480
          /* PROCEDURE FOR BEGIN,DO, ENTRY,                                            00005490
              PROC STATEMENTS    */                                                    00005500
BLOCK:      PROC (CMND);                                                               00005510
              DCL CMND CHAR(*) VARYING;                                                00005520                 09/02/74
          /* ENTRY SKIPS LINE, NO LEVEL CHANGE    */                                   00005530
          IF (CMND = 'ENTRY') THEN                                                     00005540
                DO;                                                                    00005550
                  SKIP(RINDEX) = 1;                                                    00005560
              FLAG = '0'B;
            END;
      ELSE
            DO;
              SKIP(RINDEX) = 0;
              FLAG = '1'B;
            END;
      /* ENTER IN STACK WITH LABEL(S)    */
      CALL PUSHDON (CMND,LABEL(RINDEX,*));
      TYPE(RINDEX) = LTYPE(4);
      LEVEL(RINDEX) = CLEVEL;
      CALL STEXT(LINE);
      END BLOCK;
      /* THIS ROUTINE STORES COMMENT STATEMENTS    */
COMMENT:    PROC;
      TYPE(RINDEX) = LTYPE(7);
      /* ALL COMMENTS ARE LEVEL 0    */
      LEVEL(RINDEX) = 0;
      IF (COUNT = 1) THEN GO TO CSKIP;
      /* GROUP COMMENTS, SPACE BEFORE FIRST
          AND AFTER LAST    */
      IF TYPE(LINDEX(COUNT-ONEB)) = LTYPE(7)
            THEN
            DO;
               PT = SUBPTR(RINDEX);
               IF (SKIP(PT) = TWOB) THEN
                  SKIP (PT) = ONEB;
            ELSE
               SKIP (PT) = ZB;
            SKIP(RINDEX) = 3;
        END;
            ELSE
CSKIP:            SKIP(RINDEX) = TWOB;
            CALL STEXT(LINE);
            END COMMENT;
      /* CGFIND ( OR CGEND ) SEARCHES BUFFER FOR
          CONCEPTUAL GROUPS    */
CGFIND:     PROC (NST,NEND);
      DCL (LCT(3),LT,KC,JC,L2END,TEND,JEND)
            BIN FIXED(15,0),
            STYPE CHAR(4) VAR,
            FTYPE CHAR(4) VAR INIT ('');
      /* CONSIDER EACH GROUP OF HALFCG
          STATEMENTS, STARTING AT TOP OF BUFFER
          AND CONTINUING UNTIL BUFFER BOTTOM
          -1 IS HIT. IF TESTCG
          STATEMENTS OF A GROUP ARE OF ONE
          TYPE (ASSIGNMENT, IO. OR CALL),
          RETURN NST = # OF FIRST CG ST.,
          NEND = # OF LAST CG ST.     */
      JEND = HALFCG;
      TEND = COUNT - ONEB;
LO:   NST,NEND = ZB;
```

```
    L1:   DO  JC = ONEB TO JEND;
          LCT = ZB;
          L2END = JC + HALFCG - ONEB;
    L2:   DO KC = JC TO L2END;
          STYPE = TYPE(LINDEX(KC));
          /* CENTERS FOR CALL, IO, ASSIGN ST.    */
          DO LT = 1 TO 3;
                IF LTYPE(LT) = STYPE THEN
                  DO;
                  LCT(LT) = LCT(LT) + ONEB;
                  IF LCT(LT) = TESTCG THEN
                  DO;
                  FTYPE = LTYPE(LT);
                  NEND = KC;
                GO TO L3;
                END;
          END  L2;
          /* NO CG IF NEND IS STILL 0    */
          IF (NEND = ZB) THEN
                GO TO L4;
          /* FOUND CG, DOES IT EXTEND FURTHER ?    */
L3: IF (NEND=TEND) THEN
                GO TO FIRST;
          DO KC = NEND + ONEB TO TEND;
             IF TYPE (LINDEX(KC)) = FTYPE THEN
                      NEND = KC;
             ELSE
                GO TO FIRST;
          END;
          /* FIND FIRST CG-TYPE STATEMENT    */
    FIRST:      DO KC = JC TO JC + HALFCG - TESTCG;
                IF TYPE(LINDEX(KC)) = FTYPE THEN
                  DO;
                  NST = KC;
                  RETURN;
                  END;
    L4:   END  L1;
          /* NO CG, NST AND NEND STILL 0    */
          RETURN;
          /* ENTER HERE TO SEARCH FOR CG IN
              PARTIALLY FILLED BUFFER AT END    */
    CGEND:     ENTRY(NST,NEND);
          JEND = COUNT + ONEB - HALFCG;
          TEND = COUNT;
                GO TO L0;
                END CGFIND;
%PAGE;
/* ********* PAGE 36 IN HANDWRITTEN COPY ********* */


          /* PROCESSES CALL, GO TO, EXIT, STOP, WAIT,
              DELAY, RETURN STATEMENTS    */
    CTRLC:      PROC;
```

```
          /* ENTRY FOR CALL   */
          TYPE(RINDEX) = LTYPE(3):
          IF (CCUNT = ONEB) THEN GO TO ONC1:
          /* FIND PRECEDING LINE SKIPCODE. NO
              SKIP BETWEEN SUCCESSIVE CALLS   */
          PT = SUBPTR(RINDEX):
          IF (SKIP(PT) = 5) & (TYPE(LINDEX(COUNT - ONEB))
                = LTYPE(3)) THEN
                SKIP(PT) = ZB:
    ONC1:       SKIP(RINDEX) = 5:
ONC2:       IF (PLEV¬=0) THEN
                  DO:
                LEVEL(RINDEX) = PLEV:
                PLEV = 0:
                END:
          ELSE
             LEVEL(RINDEX)=CLEVEL:
          CALL STEXT(LINE):
          RETURN:
          /* ENTRY FOR GO TO ST.   */
    CTRLG:      ENTRY:
          TYPE(RINDEX) = LTYPE(4):
          GO TO ONC1:
          /* ENTRY FOR ALL OTHER CONTROL ST.   */
    CTRLO:      ENTRY:
          TYPE(RINDEX) = LTYPE(4):
          SKIP(RINDEX) = 3:
            GO TO ONC2:
          END CTRLC:
    DECLARE:    PROC:
          /* FOR DECLARATIONS, ALLOCATE AND DEFAULT ST.   */
          DCL (NCHAR,L) BIN FIXED(15,0),
                SLEV DEC FIXED(2):
          /* FIRST LINE OF STATEMENT   */
          LEVEL(RINDEX) = CLEVEL:
          TYPE(RINDEX) = LTYPE(5):
          SKIP(RINDEX) = ONEB:
          NCHAR = 5:
          /* SUBST 'DCL' FOR FULL WORD   */
          IF (SUBSTR(LINE,1,7) = 'DECLARE') THEN
                LINE = 'DCL' || SUBSTR(LINE,8):
          /* SEPARATE PHASES : FIND FIRST COMMA
              NOT IN QUOTE OR PARENS   */
    FCOMMA:     CALL FINDCOM(L):
          TEMLINE = SUBSTR(LINE,1,L):
          /* FIRST LINE STARTS AT CURRENT MATGIN   */
          IF (NCHAR = 5) THEN
                GO TO PUT1:
          /* FOR OTHER LINES   */
          /* FIND MARGIN, PREPARE TO STORE   */
         IF SUBSTR(TEMLINE,1,2)=BRK(6) THEN DO:                        09/02/74
              TLINE=LINE:                                              09/02/74
              L=INDEX(TEMLINE,BRK(7))+1:                               09/02/74
              LINE=SUBSTR(TEMLINE,1,L);                                       00007124          09/02/74
              CALL COMMENT;                                                   00007125          09/02/74
              LINE=TLINE;                                                     00007126          09/02/74
              GO TO PUT1A;                                                    00007127          09/02/74
         END;                                                                 00007128          09/02/74
          IF (VERIFY(SUBSTR(TEMLINE,1,1),'0123456789')                        00007130
                = ZB) THEN                                                    00007140
              SLEV=DEC(SUBSTR(TEMLINE,1,INDEX(TEMLINE,' ')-1),2,0);           00007150          09/02/74
          ELSE                                                                00007160
                SLEV = 1;                                                     00007170
          LEVEL(RINDEX) = CLEVEL + SLEV;                                      00007180
          SKIP(RINDEX) = ZB;                                                  00007190
          PREFIX(RINDEX,*)='';                                                00007200          09/02/74
          TYPE(RINDEX)='';                                                    00007201          09/02/74
          LABEL(RINDEX,*)='';                                                 00007202          09/02/74
          /* STORE PHASE IN BUFFER   */                                       00007220
    PUT1:       CALL STEXT (TEMLINE);                                         00007230
          /* MOVE LINE UP OPERATE ON NEXT PART    */                          00007240
    PUT1A:                                                                    00007241          09/02/74
          LINE = SUBSTR(LINE,L + 1);                                          00007250
          /* DELETE LEADING BLANKS    */                                      00007260
         IF VERIFY(LINE,BLANK)¬=0 THEN                                        00007270
                LINE = SUBSTR(LINE,VERIFY(LINE,BLANK));                       00007280
          /* FINISHED ?    */                                                 00007290
          IF LINE = '' THEN                                                   00007300
                GO TO DCLEND;                                                 00007310
          /* NO, REPEAT    */                                                 00007320
          NCHAR = ONEB;                                                       00007330
          RINDEX = ADDPTR(RINDEX);                                            00007340
                GO TO FCOMMA;                                                 00007350
    DCLEND:     IF SKIP(RINDEX) = ONEB THEN                                   00007360
                SKIP(RINDEX) = TWOB;                                          00007370
          ELSE                                                                00007380
                SKIP(RINDEX) = 3;                                             00007390
          END   DECLARE;                                                      00007400
          /* ELSE IS CALLED BY PUSHPUL    */                                  00007410
    ELSE:       PROC;                                                         00007420
          TYPE(RINDEX) = LTYPE(4);                                            00007440
          SKIP(RINDEX) = ZB;                                                  00007450
    /* POP UP USED UP IFS */                                                  00007451          09/02/74
        DO WHILE(STACK1='ELSE');                                              00007452          09/02/74
              CALL POPUP('ELSE');                                             00007453          09/02/74
              CLEVEL=STACK3-1;                                                00007454          09/02/74
              CALL POPIF;                                                     00007455          09/02/74
          END;                                                                00007456          09/02/74
          /* CHECK FOR MATCHING IF    */                                      00007460
/* GB  */                                                                     00007470          09/02/74
  IF STACK1¬='IF' THEN DO;                                                    00007471          09/02/74
      PUT SKIP LIST('**ERROR IN IF...ELSE STRUCTURE**');                      00007472          09/02/74
              STOP;                                                           00007473          09/02/74
END;                                                                          00007474          09/02/74
 /* END OF GB  */                                                             00007475          09/02/74
          /* ENTER IN PUSHDOWN LIST WITH NULL LABEL    */                     00007490
         FLAG='C'B;                                                 00007500          09/02/74
         CALL PUSHDON('ELSE',MTLAB);                                00007501          09/02/74
         CLEVEL=STACK3;                                             00007502          09/02/74
LEVEL(RINDEX)=CLEVEL-1;                                             00007503          09/02/74
       /* IS ELSE FOLLOWED BY SEMICOLON (EMPTY)    */               00007520
       TEMLINE = SUBSTR(LINE,5);                                    00007530
       TEMLINE = SUBSTR(TEMLINE,VERIFY(TEMLINE,BLANK));             00007540
       IF  (SUBSTR(TEMLINE,1,1) = BRK(3)) THEN                      00007550
          DO;                                                       00007560
             CALL STEXT(LINE);                                      00007570
             THFL = TWOB;                                           00007580
             RETURN;                                                00007600
          END;                                                      00007610
       /* ELSE IS FOLLOWED BY TEXT, SEPARATE                        00007620
           ELSE   */                                                00007630
       CALL STEXT('ELSE');                                          00007640
       /* REMAINING TEXT, NEW LINE   */                             00007650
      IF SUBSTR(TEMLINE,1,2)=BRK(6) THEN DO;                        00007651          09/02/74
         RINDEX=ADDPTR(RINDEX);                                     00007652          09/02/74
         TLINE=LINE;                                                00007653          09/02/74
          L=INDEX(TEMLINE,BRK(7))+1;                                00007654          09/02/74
        LINE=SUBSTR(TEMLINE,1,L);                                   00007655          09/02/74
         CALL COMMENT;                                              00007656          09/02/74
        LINE=TLINE;                                                 00007657          09/02/74
         TEMLINE=SUBSTR(TEMLINE,L+1);                               00007658          09/02/74
         TEMLINE=SUBSTR(TEMLINE,VERIFY(TEMLINE,BLANK));             00007659          09/02/74
      END;                                                          00007660          09/02/74
       RINDEX = ADDPTR(RINDEX);                                     00007661
              COUNT=COUNT+ONEB;                                     00007662          09/02/74
               LINDEX(COUNT)=RINDEX;                                00007663          09/02/74
       LINE = TEMLINE;                                              00007670
       /* ANY PREFIX ?   */                                         00007680
       CALL RLABEL;                                                 00007690
       /* IS THIS ON IF OR DO STATEMENT    */                       00007700
       CMND = SUBSTR(LINE,1,3);                                     00007710
            IF CMND='BEGIN ' | CMND='BEGIN;' |                      00007720          09/02/74
             CMND='DO ' | CMND='DO;' THEN CLEVEL=CLEVEL-1;          00007721          09/02/74
       END ELSE;                                                    00007810
 FINDCOM:    PROC(LL);                                              00007820
       /* EXAMINES LINE,RETURNS LL = POSTION OF                     00007830
           FIRST COMMA NOT IN QUOTES OR PARENS.                     00007840
           IF NONE, LL = LENGTH (LINE).   */                        00007850
       DCL (KC,PAREN,LL) BIN FIXED(15,0),                           00007860
           QUOTE  BIT(1) INIT('0'B),                                00007870
           KCHAR CHAR;                                              00007880
       PAREN = ZB;                                                  00007890
 L1:   DO KC = 1 TO LENGTH(LINE) - 1;                               00007900
       KCHAR = SUBSTR(LINE,KC,1);                                   00007910
       /* DONT LOOK INSIDE QUOTES OR PARENS   */                    00007920
       IF (KCHAR = BRK(2)) & (PAREN = ZB) & (¬QUOTE)                00007930
             THEN                                                   00007940
             GO TO FCOM;                                            00007950
       /* IS IT QUOTE OR PAREN ?   */                               00007960
      IF (KCHAR = BRK(1)) THEN                                     00007970
            DO;                                                    00007980
            QUOTE = ¬QUOTE;                                        00007990
            GO TO L3;                                              00008000
      END;                                                         00008010
      IF (KCHAR = BRK(4)) THEN                                     00008020
            PAREN = PAREN + ONEB;                                  00008030
      ELSE                                                         00008040
            IF (KCHAR = BRK(5)) THEN                               00008050
              DO;                                                  00008060
              PAREN = PAREN - ONEB;                                00008070
              IF PAREN < ZB THEN                                   00008080
                GO TO PERROR;                                      00008090
            END;                                                   00008100
L3:   END L1;                                                      00008110
      IF (QUOTE) | (PAREN ¬= 0) THEN                               00008120
            GO TO PERROR;                                          00008130
L4:   LL = LENGTH(LINE);                                           00008140
      RETURN;                                                      00008150
      /* FOUND COMMA    */                                         00008160
FCOM:       LL = KC;                                               00008170
            RETURN;                                                00008180
PERROR:     PUT SKIP LIST ('**UNBALANCED PARENS OR QUOTES**');     00008190
      GO TO L4;                                                    00008200
      END  FINDCOM;                                                00008210
IF:   PROC;                                                        00008220
      DCL  L BIN FIXED(15,0);                                      00008230
      /* FOR EACH IF CLAUSE    */                                  00008240
      FLAG = '1'B;                                                 00008250
OVER:       TYPE(RINDEX) = LTYPE(4);                               00008260
      SKIP(RINDEX) = ZB;                                           00008270
      /* ENTER IN PUSHDOWN LIST, NULL LABEL    */                  00008280
      CALL PUSHDON ('IF',MTLAB);                                   00008290
 LEVEL(RINDEX)=CLEVEL-1;                                           00008300          09/02/74
      /* SEPARATE FIRST PHASE THRU THEN    */                      00008310
      L = INDEX(LINE,' THEN ');                                    00008320
      IF L = ZB THEN                                               00008330
        DO;                                                        00008340
            L = INDEX(LINE,' THEN;');                              00008350
            IF L = ZB THEN                                         00008360
                  GO TO THENERR;                                   00008370
      /* FOUND 'THEN;', EMPTY CLAUSE    */                         00008380
            ELSE                                                   00008390
              DO;                                                  00008400
              TLINE=SUBSTR(LINE,1,L+5);                            00008410          09/02/74
              CALL STEXT(TLINE);                                   00008411          09/02/74
          GO TO TESTFN;                                            00008412          09/02/74
              END;                                                 00008430
                END;                                               00008431          09/02/74
      /* FOUND 'THEN ' */                                          00008440
         TLINE=SUBSTR(LINE,1,L+4);                                 00008450          09/02/74
         CALL STEXT(TLINE);                                        00008451          09/02/74
      /* UPDATE PTR, LINE, DELETE LEADING BLANKS    */             00008460
```

227

Moore Business Forms, Inc

```
      LINE = SUBSTR(LINE,L + 6):                                 00008470
      LINE = SUBSTR(LINE,VERIFY(LINE,BLANK)):                    00008480
    IF SUBSTR(LINE,1,2)=BRK(6) THEN DO:                          00008481            09/02/74
          RINDEX=ADDPTR(RINDEX):                                 00008482            09/02/74
          TLINE=LINE:                                            00008483            09/02/74
          L=INDEX(TLINE,BRK(7))+1:                               00008484            09/02/74
          LINE=SUBSTR(TLINE,1,L):                                00008485            09/02/74
          CALL COMMENT:                                          00008486            09/02/74
          LINE=SUBSTR(TLINE,L+1):                                00008487            09/02/74
          LINE=SUBSTR(LINE,VERIFY(LINE,BLANK)):                  00008488            09/02/74
     END:                                                        00008489            09/02/74
      RINDEX = ADDPTR(RINDEX):                                   00008490
             COUNT=COUNT+ONEB:                                   00008491            09/02/74
              LINDEX(COUNT)=RINDEX:                              00008492            09/02/74
      /* CHECK FOR PREFIX FOR NEW LINE   */                      00008500
      CALL RLABEL:                                               00008510
   IF SUBSTR(LINE,1,3)='DO ' | SUBSTR(LINE,1,3)='DO:' |          00008511            09/02/74
           SUBSTR(LINE,1,6)='BEGIN ' | SUBSTR(LINE,1,6)='BEGIN:'  00008512            09/02/74
                 THEN CLEVEL=CLEVEL-1:                           00008513            09/02/74
      /* ANOTHER IF ?   */                                       00008520
      IF (SUBSTR(LINE,1,3) = 'IF ') |                            00008530
            (SUBSTR(LINE,1,3) = 'IF(') THEN                      00008540
                  GO TO OVER:                                    00008550
RET:  THFL = 1:                                                  00008560
      RETURN:                                                    00008570
      /* NO 'THEN', PRINT MESSAGE, FUDGE, GO ON   */             00008580
THENERR:    PUT SKIP LIST('**MISSING ''THEN'' IN IF STATEMENT**'): 00008590
      PUT SKIP:                                                  00008600
      CALL STEXT(LINE):                                          00008610
      /* RETURN TO MAIN PROGRAM   */                             00008620
      GO TO P3:                                                  00008630
            END IF:                                              00008640
IOSUB:      PROC:                                                00008650
      /* PROCESSES IO STATEMENTS. SKIP LINE BEFORE               00008660
          AND AFTER EACH IO OR GROUP OF IO S.   */               00008670
      TYPE(RINDEX) = LTYPE(2):                                   00008680
      IF (PLEV ¬= 0) THEN                                        00008690
            DO:                                                  00008700
                  LEVEL(RINDEX) = PLEV:                          00008710
                       PLEV = 0:                                 00008720
                  END:                                           00008730
      ELSE                                                       00008740
                  LEVEL(RINDEX) = CLEVEL:                        00008750
      /* IS PRECEDING STATEMENT ALSO IO ?   */                   00008760
      IF COUNT = 1 THEN GO TO I11:                               00008770
      IF (TYPE(LINDEX(COUNT - 1)) = LTYPE(2)) THEN               00008780
            DO:                                                  00008790
            PT = SUBPTR(RINDEX):                                 00008800
            IF SKIP(PT) = TWOB THEN                              00008810
                  SKIP(PT) = ONEB:                               00008820
            ELSE                                                 00008830
                  SKIP(PT) = ZB:                                 00008840
            SKIP(RINDEX) = 3:                                    00008850
```

228

```
                END;
          ELSE
    I11:                  SKIP(RINDEX) = TWOB;
          /* ENTER IN BUFFER    */
          CALL STEXT(LINE);
          END IOSUB;
MOVUP:    PROC(THROO);
          /* CALLED AFTER OUTPUT TO MOVE UP REMAING BUFFER
              LINES.  ACTUALLY ONLY THE LINDEX TABLE IS
              CHANGED (POINTERS TO FIRST LINE OF EACH STATEMENT
              IN BUFFER(. COUNT IS THE NUMBER OF STATEMENTS
              IN BUFFER, THROO IS THE NUMBER OF LAST ST
              TO BE MOVED OUT. UPDATES COUNT,RINDEX    */
          DCL (THROO,JCT) BIN FIXED(15,0);
                COUNT = COUNT - THROO;
                IF COUNT = ZB THEN
                   RETURN;
          DO JCT = 1 TO COUNT;
                LINDEX(JCT) = LINDEX(JCT + THROO);
          END;
          /* ZERO UNUSED INDICES    */
          DO JCT = COUNT + 1 TO 20;
                LINDEX(JCT) = ZB;
          END;
          END MOVUP;
    ONSUB:      PROC;
          /* PROCESSES ON CONDITION, SIGNAL AND REVERT
              STATEMENTS. PREDEDES
              STATEMENTS. PRECEDES EACH BY BLANK SPACE AND
              DOTTED LINE, FOLLOWS BY DOTTED LINE, UNLESS
              ON ST. INCLUDES A BEGIN GROUP, WHEN
              ONFLAG IS SET TO 1 AND THE FOLLOWING DOTTED
              LINE IS IMPLEMENTED AFTER THE CORRESPONDING
              END.    */
          DCL   LL FIXED BIN(15,0);
          TYPE(RINDEX) = LTYPE(6);
          IF (PLEV ¬= 0) THEN
                DO; LEVEL(RINDEX) = PLEV;
                      PLEV = 0;
                END;
          ELSE
                LEVEL(RINDEX) = CLEVEL;
                SKIP(RINDEX) = 7;
          /* IS LAST (NON-BLANK) WORK 'BEGIN' ?    */
                IF LENGTH(LINE) < 7 THEN
                GO TO TSTOR;
    L1:         DO LL = LENGTH(LINE) - 1 TO 6 BY -1;
          IF SUBSTR(LINE,LL,1) = BLANK THEN
                GO TO L2;
          IF SUBSTR(LINE,LL-5,6) = ' BEGIN' THEN
                DO;
                   ONFLAG = '1'B;
                   SKIP(RINDEX) = 6;
                END:                                                    00009390
          GO TO TSTOR:                                                  00009400
    L2:   END   L1:                                                     00009410
    TSTOR:      CALL STEXT(LINE):                                       00009420
          END ONSUB:                                                    00009430
    OUTPUT:     PROC(THRU):                                             00009440
          /* PRINTS STATEMENTS 1 - THRU   */                            00009450
          /* WRITTEN FOR 80 OR 120 COLUMN PRINTOUT                      00009460
              ON SYSTEM FILE   */                                       00009470
          DCL (RI,THRU, NV, LASTN, I) BIN FIXED(15,0),                  00009480
                (LEV, MC) DEC FIXED (3,0),                              00009490
                (CMTB INIT ('/* '), CMTE INIT (' */')) CHAR(3):         00009500
          IF THRU = ZB THEN                                             00009510
                RETURN:                                                 00009520
          /* ALL LINES ARE AT SAME LEVEL (RULE 1)    */                 00009530
          LEV = LEVEL(LINDEX(1)):                                       00009540
          MC = MARGIN(LEV) + 1:                                         00009550
          /* FIND POINTER TO LAST OUTPUT LINE    */                     00009560
          IF THRU = COUNT THEN                                          00009570
                LASTN = RINDEX:                                         00009580
          ELSE                                                          00009590
                      LASTN = SUBPTR(LINDEX(THRU + 1)):                 00009600
          RI = SUBPTR(LINDEX(1)):                                       00009610
          /* PRINT ONE LINE AT A TIME    */                             00009620
    LPO:        DO I = 1 TO 50:                                         00009630
          RI = ADDPTR(RI):                                              00009640
                    LEV=LEVEL(RI):                                      00009641           09/02/74
                   MC=MARGIN(LEV)+1:                                    00009642           09/02/74
          /* IS IT COMMENT ?    */                                      00009650
          IF TYPE(RI) = LTYPE(7) THEN                                   00009660
                DO:                                                     00009670
    OCMT:       PUT FILE(SYSPRINT) EDIT                                 00009680
            (CMTB,TEXT(RI),CMTE,SEQ#(RI))                               00009690           09/02/74
        (COL(2),A,A,COL(NCOL-2),A,X(4),A):                              00009691           09/02/74
          GO TO LPA:                                                    00009710
 /* GB */    END:                                                       00009711           09/02/74
          /* NOT COMMENT: IS TEXT NULL ?    */                          00009720
          IF TEXT(RI) = '' THEN                                         00009730
                DO:                                                     00009740
NTXT:     PUT FILE(SYSPRINT) EDIT(PREFIX(RI,*),                         00009750
         LABEL(RI,*),LEV,SEQ#(RI))                                      00009760           09/02/74
           (COL(2),10 A,COL(NCOL+2),F(2),X(1),A):                       00009761           09/02/74
          GO TO LPA:                                                    00009780
          END:                                                          00009790
          /* LINE HAS TEXT    */                                        00009800
    TXT:        PUT FILE(SYSPRINT) EDIT                                 00009810
                (PREFIX(RI,*), LABEL(RI,*),                             00009820
              TEXT(RI),LEV,SEQ#(RI))                                    00009830           09/02/74
                (COL(2), (10)A, COL(MC), A, COL(NCOL+2),                00009840
            F(2),X(1),A):                                               00009850           09/02/74
    LPA:        IF RI = LASTN THEN                                      00009860
                GO TO LPOUT:                                            00009870
    LPE:        END LPO:                                                00009880
```

```
          /* ALL LINES PRINTED, LOOK FOR AFTER SHIP OR DOT      */
          /* BY RULES 2 & 3, ONLY LAST LINE CAN HAVE FOLLOWING SKIP OR
              DOT    */
    LPOUT:            IF SKIP(LASTN) = 0 THEN
                RETURN;
          ELSE
                NV = SKIP(LASTN);
          /* PRINT DOT, SKIP    */
    PUNC:       IF (NV = 5) | (NV = 4) THEN
                IF (NCOL = 72) THEN
                      PUT FILE(SYSPRINT) EDIT
                      (CMTB, (65) '.', CMTE)
                      (COL(2), A, A, COL(70), A);
          ELSE
                PUT FILE(SYSPRINT) EDIT
                (CMTB, (112) '.', CMTE)
                (COL(2), A, A, COL(117), A);
          IF (NV = 5) | (NV = 3) THEN
                PUT SKIP;
          RETURN;
          /* IF VERY FIRST STATEMENT HAS PRE-SKIP
              ETC., ENTER HERE    */
    OUTFST:     ENTRY(NV2);
          IF (NV2 = 1) THEN
                NV = 3;
          ELSE
                IF (NV2 = 6) THEN
                      NV = 5;
          ELSE
                IF (NV2 = 8) THEN
                      NV = 4;
          GO TO PUNC;
          END OUTPUT;
    PEND:       PROC;
          /* PROCESSES END ST, WHICH MAY TERMINATE
              PROC, DO, OR BEGIN BLOCK. UPDATES
              PUSHDOWN LIST AND LEVEL    */
          DCL LAB CHAR (31) VAR;
          DCL LVAR LABEL   (FAN3, PLAIN1);
          DCL (LINC, IFAN) BIN FIXED(15,0);
          TYPE(RINDEX) = LTYPE(4);
          LEVEL(RINDEX) = CLEVEL;
          /* SKIP IF THIS IS END OF A CONDITION BLOCK    */
/* BEGINNING GB */                                                                       09/02/74
              IF ONFLAG THEN SKIP(RINDEX)=4;                                             09/02/74
              ELSE SKIP(RINDEX)=ZB;                                                      09/02/74
        /* STORE TEXT  */                                                                09/02/74
              CALL STEXT(LINE);                                                          09/02/74
              IF (ONFLAG) THEN DO;                                                       09/02/74
                   ONFLAG='0'B;                                                          09/02/74
                   RETURN;                                                               09/02/74
                   END;                                                                  09/02/74
 /* END OF GB */                                                                         09/02/74
```

```
          /* IS END FOLLOWED BY IDENTIFIER ?   */                         00010410
          TEMLINE = SUBSTR(LINE,4);                                       00010420
          TEMLINE = SUBSTR(TEMLINE,VERIFY(TEMLINE,BLANK));                00010430
          IF LENGTH(TEMLINE) = 1 THEN                                     00010440
                GC TO PLAIN;                                              00010450
          /* SET IDENTIFIER INTO LAB   */                                 00010460
/* GB */                                                                  00010470         09/02/74
FANCY:  LAB=SUBSTR(TEMLINE.1.LENGTH(TEMLINE)-1)||':';                     00010471         09/02/74
          LIND = INDEX(LAB,BLANK);                                        00010480
          IF LIND ¬= ZB THEN                                              00010490
  /*  GB  */  LAB=SUBSTR(LAB,1.LIND-1)||':';                              00010500         09/02/74
          /* TEST FCR ERROR IN PUSHDOWN LIST   */                         00010510
    FAN2:       LVAR = FAN3;                                              00010520
                GO TO STERR;                                              00010530
          /* RETURN HERE IF NO ERROR   */                                 00010540
          /* DOES IDENTIFIER MATCH STACK LABEL ?   */                     00010550
    FAN3:       DC IFAN = 1 TO 5;                                         00010560
                IF (STACK2(IFAN) = LAB) THEN                              00010570
                    IF (STACK1='ENTRY') THEN                              00010580
                      GO TO ENTERR;                                       00010590
          ELSE                                                            00010600
                      GO TO HAVIT;                                        00010610
          /* IDENTIFIER NOT IN THIS STACK LEVEL   */                      00010620
          END;                                                            00010630
          IF STACK1 = 'ENTRY' THEN                                        00010640
                FLAG = '0'B;                                              00010650
          ELSE                                                            00010660
                FLAG = '1'B;                                              00010670
          CALL POPUP(STACK1);                                             00010680
          /* KEEP LCOKING FOR LAB   */                                    00010690
          GO TO FAN2;                                                     00010700
          /* JUST END NO IDENTIFIER   */                                  00010710
          /* TEST FCR STACK ERROR   */                                    00010720
    PLAIN:      LVAR = PLAIN1;                                            00010730
                GO TO STERR;                                              00010740
          /* RETURN HERE IF OK    */                                      00010750
    PLAIN1:     IF STACK1 = 'ENTRY' THEN                                  00010760
                DO;                                                       00010770
                      FLAG = 'C'B;                                        00010780
                      CALL POPUP(STACK1);                                 00010790
                      GO TO PLAIN;                                        00010800
                END;                                                      00010810
          /* POPUP 1 LEVEL    */                                          00010820
    HAVIT:      FLAG = '1'B;                                              00010830
          CALL POPUP(STACK1);                                             00010840
     IF STACK1='IF' THEN CLEVEL=STACK3;                                   00010850         09/02/74
                   CLEVEL=STACK3;                                         00010851         09/02/74
                   LEVEL(RINDEX)=CLEVEL+1;                                00010852         09/02/74
          RETURN;                                                         00010870
    /* ERROR RETURNSN*/                                                   00010880
ENTERR: PUT SKIP LIST('**LABEL ON END STATEMENT SHOULD NOT MATCH '||      00010890         09/02/74
        'ENTRY NAME**');                                                  00010891         09/02/74
    /* TRY TO RECOVER*/                                                   00010910
    GO TO PLAIN1:                                                        00010920
    /* TEST FOR STACK ERROR */                                           00010930
STERR: IF STACK1='TOP' | CLEVEL=0 THEN                                   00010940          09/02/74
      DO;                                                                00010950
        PUT SKIP LIST('**UNMATCHED END OR ERROR IN IF...ELSE '||         00010960          09/02/74
              'STRUCTURE**');                                            00010961          09/02/74
      STOP;                                                              00010980
      END;                                                               00010990
      GO TO LVAR:                                                        00011000
END PEND;                                                                00011010
    POPIF:      PROC:                                                    00011020
          /* POPIF (CALLED BY PUSHPUL) OR ENTRY POPEL                    00011030
              (CALLED BY ELSE OR PEND) CLEARS PUSHDOWN                   00011040
              LSIT WHEN IF, IF ... ELSE PAIR, OR BLOCK                   00011050
              IS TERMINT                                                 00011060
              IS TERMINATED. CALLS POPUP TO DO CLEARING.   */            00011070
          DCL  ELFLAG BIT(1) INIT('0'B):                                 00011080
          /* POPUP TOP 'IF'    */                                        00011090
    POP1:       FLAG='0'B:                                               00011100          09/02/74
          CALL POPUP('IF'):                                              00011110
          /* IS LIST EMPTY ?   */                                        00011120
          END POPIF:                                                     00011370
    POPUP:      PROC(CMAND):                                             00011380
          /* POPS UP 1 STACK LEVEL (IN EACH OF THE                       00011390
              3 LISTS), DECREMENTS CLEVEL IF FLAG = 1    */              00011400
             DCL CMAND CHAR(*) VARYING:                                  00011410          09/02/74
          /* NO LEVEL CHANGE IF ENTRY OR ELSE    */                      00011420
          IF (FLAG) THEN                                                 00011430
                CLEVEL = CLEVEL - 1:                                     00011440
      IF (STACK1 ¬= CMAND) THEN                                          00011450
                DO:                                                      00011460
                      PUT SKIP LIST ('**ERROR IN STACK1**'):             00011470
                      STOP:                                              00011480
                      END:                                               00011490
          /* POPUP 1 LEVEL IN EACH STACK    */                           00011500
          FREE STACK1,STACK2,STACK3:                                     00011510
          END POPUP:                                                     00011610
    PUSHDON:    PROC(CMAND,LABL):                                        00011620
          /* ENTERS OP. NAME (E.G. 'IF') IN                              00011630
              STACK1, UP TO 5 LABELS IN                                  00011640
              STACK2, CLEVEL IN STACK3.                                  00011650
              STORAGE FOR THE STACKS IS CONTROLLED,                      00011660
              ALLOCATED IN PUSHDON, FREED IN POPUP   */                  00011670
            DCL CMAND CHAR(*) VAR,                                       00011680          09/02/74
                  LABL(5) CHAR(*) VAR,                                   00011681          09/02/74
              X DEC FIXED(2):                                            00011700
          /* GET PRESENT LEVEL    */                                     00011710
                X = STACK3:                                              00011720
     /*   INCREMENT LEVEL UNLESS ENTRY, ELSE, OR IF */                   00011730          09/02/74
          IF (FLAG) THEN                                                 00011740
                X = X + 1:                                               00011750
          /* PUSHDOWN EACH OF 3 STACKS    */                             00011760
          ALLOCATE STACK1, STACK2(5), STACK3:                            00011770
```

233

```
                STACK3 = X:
                STACK1 = CMAND:
                STACK2 = LABL:
          /* UPDATE CURRENT LEVEL   */
                CLEVEL = STACK3:
          END PUSHCON:
    PUSHPUL:    PROC:
          /* CALLED BY PLEDIT OR RETURN FROM READ TO CHECK ON
              PRESENT STATUS OF IF...ELSE STRUCTURES AND DO ANY
              NECESSARY UPDATING OF THE PUSHDOWN STOCKS   */
          DCL  L BIN FIXED(15,C):
          THFL = C:
          /* ELSE STATEMENT ?   */
          IF (SUBSTR(LINE,1,5) = 'ELSE ') | (SUBSTR
              (LINE,1,5) = 'ELSE:') THEN
              DO:
                CALL ELSE:
              IF THFL=TWOB THEN GO TO P3:                        09/02/74
                RETURN:
          END:
          /* NO: IS TOP OF STACK 'IF' ?   */
AGIN1: IF STACK1='ELSE' THEN CALL POPUP('ELSE'):                 09/02/74
 IF STACK1='IF' THEN DO:                                         09/02/74
              DO WHILE(STACK1='IF'):                             09/02/74
          CLEVEL=STACK3-1:                                       09/02/74
         CALL POPIF:                                             09/02/74
           END:                                                  09/02/74
        GO TO AGIN1:                                             09/02/74
        END:                                                     09/02/74
          RETURN:
          /* YES, IF ST. IS COMPLETED, CHECK OFF   */
          END PUSHPUL:
    SMARG:      PROC:
          /* GIVEN READ-IN VALUES OF IMARGIN (INITIAL
              MARGIN) AND DELMARG (MARGIN INCREMENT) THIS
              SETS UP A TABLE OF MARGIN VALUES FOR NESTING
              LEVELS 1 - 9. COMMENT STATEMENTS ARE LEVEL 0
              AND  0 MARGIN IS DEFINED AS 1   */
          DCL IMA FIXED BIN(15,0):
          MARGIN(1) = IMARGIN:
        DO IMA=2 TO 15:                                          09/02/74
          MARGIN (IMA) = MARGIN (IMA - 1) + DELMARG:
          END:
          END SMARG:
    STEXT:      PROC (SOMLIN):
          /* WRITTEN FOR 80 OR 120 COLUMN PRINTOUT (NCOL = 72
             OR 116). STORES STATEMENT TEXT IN BUFFER IN
             PRINT LINE QUANTA, ALLOWING FOR PREFIXES AND
             LABELS. DIVIDES TEXT AT WORD ENDS IF POSSIBLE
              SEPARATES FORE-AND-AFT PARTS OF SKIPCODE AND
              STORES THEM APPROPRIATELY.  */
          DCL (CFL,COMFL) BIT(1) INIT('0'B):
          DCL (LP,LL,CC) BIN FIXED(15,0) INIT(ZB):
```

234

```
      DCL (NCHAR, MG, RC, NI, FC, WSKIP) FIXED BIN(15,0),             00012240
            TLEV DEC FIXED(2),                                        00012250
     SOMLIN CHAR(*) VARYING;                                          00012260            09/02/74
    SEQ#(RINDEX)=SEQNO;                                               00012261            09/02/74
      TLEV = LEVEL(RINDEX);                                           00012270
      MG = MARGIN(TLEV);                                              00012280
      /* SEPARATE SKIPCODE INTO FOR/AFT COMPONENTS    */              00012290
      WSKIP = SKIP(RINDEX);                                           00012300
      FC = CHCODE(WSKIP,1);                                           00012310
      RC = CHCODE(WSKIP,2);                                           00012320
      /* COMMENT HAS DIFF'T PUNCT., NO PREFIX    */                   00012330
      IF TYPE(RINDEX) = LTYPE(7) THEN                                 00012340
            DO;                                                       00012350
            NCHAR = NCOL - 7;                                         00012360
            COMFL = '1'B;                                             00012370
                  GO TO AGAIN;                                        00012380
                  END;                                                00012390
      /* NOT COMMENT    */                                            00012400
      NCHAR = NCOL - MG;                                              00012410
      DO NI = 1 TO 5;                                                 00012420
            LP = LP + LENGTH(PREFIX(RINDEX,NI));                      00012430
            LL = LL + LENGTH(LABEL(RINDEX,NI));                       00012440
         END;                                                         00012450
      /* DO PREFIX, LABEL NEED SEPARATE LINE(S) ?     */              00012460
      IF LP + LL ¬> MG THEN                                           00012470
            GO TO AGAIN;                                              00012480
      /* YES, TOO LONG    */                                          00012490
      IF LP = ZB THEN                                                 00012500
            GO TO A2;                                                 00012510
      /* PREFIX PRESENT: GIVE IT A LINE    */                         00012520
      CC = CC + ONEB;                                                 00012530
      SKIP(RINDEX) = FC;                                              00012540
      PT = ACCPTR(RINDEX);                                            00012550
      LABEL(RINDEX,*)='';                                             00012560            09/02/74
      TEXT(RINDEX)='';                                                00012561            09/02/74
            RINDEX = PT;                                              00012570
    SEQ#(RINDEX)=SEQNO;                                               00012571            09/02/74
      /* NOW LABEL, IF ANY    */                                      00012580
      /* SHORT ENOUGH TO FIT IN MARGIN ?    */                        00012590
      IF LL  <= MG THEN                                               00012600
            DO;                                                       00012610
            CFL = '1'B;                                               00012620
            GO TO AGAIN;                                              00012630
            END;                                                      00012640
      /* LONGER, SEPARATE    */                                       00012650
A2:         CC = CC + ONEB;                                           00012660
      IF CC = ONEB THEN                                               00012670
            SKIP(RINDEX) = FC;                                        00012680
      ELSE                                                            00012690
            DO;                                                       00012700
            SKIP(RINDEX) = 0;                                         00012710
           PREFIX(RINDEX,*)='';                                       00012720            09/02/74
           TYPE(RINDEX)='';                                           00012721            09/02/74
      END:                                                         00012730
TEXT(RINDEX)='':                                                   00012740
      LEVEL(RINDEX) = TLEV:                                        00012750
      RINDEX = ADDPTR(RINDEX):                                     00012760
    SEQ#(RINDEX)=SEQNO:                                            00012761            09/02/74
      /* SEPARATE TEXT INTO PRINT LINES. STORE   */                00012770
AGAIN:      CC = CC + ONEB:                                        00012780
      /* CASE: PR + LABEL < MARGIN   */                            00012790
      IF CC = ONEB THEN                                            00012800
            SKIP(RINDEX) = FC:                                     00012810
      /* BUT GENERALLY:   */                                       00012820
      ELSE                                                         00012830
            DO:                                                    00012840
                  SKIP(RINDEX) = 0:                                00012850
                  IF (CFL) THEN                                    00012860
                    CFL = ¬CFL:                                    00012870
            ELSE                                                   00012880
                  LABEL(RINDEX,*) = '':                            00012890
            PREFIX(RINDEX,*) = '':                                 00012900
            IF (¬COMFL) THEN                                       00012910
                  TYPE(RINDEX) = '':                               00012920
            ELSE                                                   00012930
                  TYPE(RINDEX) = LTYPE(7):                         00012940
            END:                                                   00012950
      LEVEL(RINDEX) = TLEV:                                        00012960
      /* WILL TEXT FIT IN THIS LINE ?   */                         00012970
IF LENGTH(SOMLIN) <= NCHAR THEN                                    00012980
            DO:                                                    00012990
            TEXT(RINDEX) = SOMLIN:                                 00013000
            GO TO TEXTOUT:                                         00013010
      END:                                                         00013020
      /* NO, SEPARATE BETWEEN WORDS   */                           00013030
      DO NI = NCHAR TO 1 BY -1:                                    00013040
      IF (SUBSTR(SOMLIN,NI,1) = BLANK) THEN                        00013050
            GO TO LINOUT:                                          00013060
      END:                                                         00013070
      /* NO BLANK FOUND   */                                       00013080
      NI = NCHAR:                                                  00013090
LINOUT:     TEXT(RINDEX) = SUBSTR(SOMLIN,1,NI):                    00013100
      SOMLIN = SUBSTR(SOMLIN,NI + 1):                              00013110
      IF SOMLIN = '' THEN                                          00013120
            GO TO TEXTOUT:                                         00013130
      RINDEX = ADDPTR(RINDEX):                                     00013140
    SEQ#(RINDEX)=SEQNO:                                            00013141            09/02/74
            GO TO AGAIN:                                           00013150
      /* TEXT COMPLETE, CORRECT SKIP FOR 'AFTER'   */              00013160
TEXTOUT:    IF CC = ONEB THEN                                      00013170
            SKIP(RINDEX) = WSKIP:                                  00013180
      ELSE                                                         00013190
            SKIP(RINDEX) = RC:                                     00013200
      /* RETURN   */                                               00013210
      END STEXT:                                                   00013220
      /* ALL SUBROUTINES AND FUNCTIONS HAVE BEEN INCLUDED   */     00013230
```

```
                  END PLECIT:                                                     00013240
```

237

APPENDIX C

CONCEPTUAL GROUPINGS PROGRAM FOR PL/1 (GP-P)

6. SOURCE PL/1 PROGRAM OF (GP-P) GROUPED

```
PLEDIT: PROCEDURE OPTIONS(MAIN):                                                                                 1 00000010
        DCL TLINE VAR CONTROLLED:                                                                                1 00000011
        DCL STATEMENT_SIZE FIXED DECIMAL(7):                                                                     1 00000012
        DCL MAX_LINES FIXED DECIMAL(7):                                                                          1 00000013
        DCL SEQNO CHAR(8):                                                                                       1 00000014
        DCL CONT_CHAR CHAR(1):                                                                                   1 00000015
        DCL SYSPRINT PRINT ENV(MEDIUM(SYSLST.1403) F RECSIZE(133)  BUFFERS(2)):                                  1 00000016
        DCL CCIN STREAM INPUT ENV(MEDIUM(SYS004.2540) F RECSIZE(80)):                                            1 00000018
/* /* PART 1 INITIALIZATION */                                                                               */    00000020
/* /* BUFFER STORAGE */                                                                                      */    00000030
/* /* LINDEX IS POINTER TABLE TO BUFFER */                                                                   */    00000040
/* /* BUFFER IS 6 ARRAYS. EACH DIMENSIONED 50 */                                                             */    00000050
/* /* LABEL PREFIX ALLOW MAX OF 5 TO A ST. */                                                                */    00000060
        DCL LINDEX(2C) FIXED BIN(15.0) INIT((20)0).                                                              1 00000070
             TEXT(*) CHAR(120) VAR CONTROLLED.                                                                   2 00000070
             PREFIX(*,*) CHAR(31) VAR CONTROLLED.                                                                2 00000070
             LABEL(*,*) CHAR(31) VAR CONTROLLED.                                                                 2 00000070
             TYPE(*) CHAR(4) VAR CONTROLLED.                                                                     2 00000070
             LEVEL(*) DEC FIXED(2) CONTROLLED.                                                                   2 00000070
             SEQ#(*) CHAR(8) CONTRCLLED.                                                                         2 00000070
             SKIP(*) BIN FIXED CCNTROLLED:                                                                       2 00000070
/* /* COUNT = NUMBER OF CURRENT STATEMENT.  RINDEX IS POINTER TO CURRENT BUFFER  LINE       */                 */    00000140
        DCL(COUNT,RINDEX) BIN FIXED(15.0)INIT(0):                                                                1 00000170
/* /* MARGIN TABLE ANS PARAMETERS FOR PRINTING  AND COMPUTING MARGINS   */                                   */    00000180
        DCL MARGIN(0:15) BIN FIXED(15.0)INIT(1).                                                                 1 00000200
             (DELMARG INIT(2). IMARGIN INIT(9).  NCOL INIT(80)) BIN FIXED(15.0).                                 2 00000200
             PAGENO DEC FIXED(3.0) INIT(1):                                                                      2 00000200
/* /* PUSHDOWN LISTS : STACK1 = OPERATION.  STACK2 HAS LABEL(S). STACK3 = LEVEL    */                        */    00000240
        DCL (STACK1 CHAR(5) VAR.  STACK2(5) CHAR(31) VAR.  STACK3 DEC FIXED(2)) CONTROLLED:                      1 00000260
/* /* 80-COL RECORDS ARE READ. 1 AT A TIME. INTO  NCARD. EACH STATEMENT. AS IT IS SEPARATED  FROM NCARD. IS  */    00000290
/* STORED IN LINE FOR ANALYSIS.  TEMLINE IS WORKING STRING STORAGE    */                                     */    00000290
        DCL NCARD VAR CONTROLLED.                                                                                1 00000330
             LINE VAR CONTROLLED.                                                                                2 00000330
             TEMLINE VAR CONTROLLED:                                                                             2 00000330
/* /* MISC. VARIABLES : FLAGS. ETC.   */                                                                     */    00000360
        DCL (COMFLG.THFL) FIXED BIN(15.0) INIT(0).                                                               1 00000370
             CMND CHAR(10) VAR.                                                                                  2 00000370
             (NV1.NV2.NST.NEND.THISN.LASTN.PT)  BIN FIXED(15.0):                                                 2 00000370
        DCL (FLAG.FINIS.LEVF.ONFLAG.RFLAG)  BIT(1) INIT('0'B):                                                   1 00000410
/* /* CLEVEL IS CURRENT LEVEL. PLEV IS  SAVED PRECEDING LEVEL USED IN SOME  ELSE STATEMENTS   */             */    00000430
        DCL (CLEVEL.PLEV) DEC FIXED(2) INIT(0):                                                                  1 00000460
/* /* BINARY CONSTANTS   */                                                                                  */    00000470
        DCL (ZB INIT(0). ONEB INIT(1).  TWOB INIT(2)) BIN FIXED(15.0):                                           1 00000480
/* /* LOOK FOR CG WHEN COUNT = BUFCG(.=20).  TESTCG = NUMBER DEFINING A CG ( .=HALFCG.  WHICH IS BUFCG/2)   */  */    00000500
        DCL (BUFCG INIT(20).HALFCG.  TESTCG INIT(7)) BIN FIXED(15.0):                                            1 00000530
/* /* CHCODE TABLE SEPARATES SKIPCODE INTO FORE  AND AFT COMPONENTS   */                                     */    00000550
        DCL CHCODE(0:9.2) BIN FIXED(15.0)  INIT( 0.0. 1.0. 1.3. 0.3. 0.4. 0.5.  6.0. 6.4. 8.0. 8.4):             1 00000570
/* /* STATEMENT TYPES   */                                                                                   */    00000600
        DCL LTYPE(7) CHAR(4) VAR INIT  ('AST'. 'IO'. 'CAL'. 'CTRL'. 'STOR'.  'ON'. 'COM'):                       1 00000610
/* /* BREAK CHARATERS AND NULL ARRAY   */                                                                    */    00000640
        DCL BLANK CHAR(1) INIT(' '):                                                                             1 00000650
        DCL BRK(8) CHAR(2) VAR INIT  ('''', '.', ':', '(', ')', '/*', '*/', ':'):                                1 00000660
        DCL MTLAB(5) CHAR(2) VAR INIT((5)''):                                                                    1 00000680
/* /* INPUT IS SYSTEM FILE. CARDS.80COL.  PROVIDE FOR END OF FILE   */                                       */    00000690
/* ..................................................................................................... */
        ON ENDFILE (SYSIN)  BEGIN:                                                                               1 00000710
        FINIS = '1'B:                                                                                            1 00000730
        GO TO ENDRD:                                                                                              1 00000740
/* ............................................................................................................... */
        END:                                                                                                      1 00000750
/* ............................................................................................................... */
/* /* READ IN LINESIZE(80 OR 120) FOR OUTPUT,  MARGIN PARAMETERS    */                                          */     00000760
/* ............................................................................................................... */
        ON ENDFILE(CCIN) BEGIN:                                                                                   1 00000780
        PUT EDIT('NOT ENOUGH CONTROL INFORMATION SUPPLIED (SYS004).')(A):                                         1 00000781
        PUT EDIT('ITEM 1: LENGTH OF PRINTLINE: SUGGESTED 120')(SKIP(1),A):                                        1 00000782
        PUT EDIT('ITEM 2: BEGINNING MARGIN: SUGGESTED 9')(SKIP(1),A):                                             1 00000783
        PUT EDIT('ITEM 3: MARGIN STEP SIZE: SUGGEST 5')(SKIP(1),A):                                               1 00000784
        PUT EDIT('ITEM 4: THE NUMBER 20')(SKIP(1),A):                                                             1 00000785
        PUT EDIT('ITEM 5: THE NUMBER 7')(SKIP(1),A):                                                              1 00000786
        PUT EDIT('ITEM 6: MAX # CHARACTERS PER PL/I STATEMENT: SUGGESTED 800') (SKIP(1),A):                       1 00000787
        PUT EDIT('ITEM 7: MAX # EDITED LINES PER PL/I STATEMENT: SUGGESTED '|| '50')(SKIP(1),A):                  1 00000789
        PUT EDIT('PROGRAM TERMINATED.')(SKIP(1),A):                                                               1 00000791
/* ............................................................................................................... */
        SIGNAL ERROR:                                                                                             1 00000792
/* ............................................................................................................... */
        END:                                                                                                      1 00000793
/* ............................................................................................................... */
        OPEN FILE(CCIN),FILE(SYSIN):                                                                              1 00000794
        GET FILE(CCIN) LIST(NCOL,IMARGIN,DELMARG):                                                                1 00000795
/* /* DEFINE OPTIONS FOR OUTPUT (SYSPRINT)    */                                                                */     00000796
        OPEN FILE(SYSPRINT):                                                                                      1 00000800
/* /* ACTION AT PAGE END    */                                                                                  */     00000820
/* ............................................................................................................... */
        ON ENDPAGE (SYSPRINT)  BEGIN:                                                                             1 00000830
        PAGENO = PAGENO+1:                                                                                        1 00000850
        PUT FILE (SYSPRINT) PAGE EDIT  ('PAGE ', PAGENO)(COL(NCOL-8),A,F(3)):                                     1 00000860
        END:                                                                                                      1 00000880
/* ............................................................................................................... */
/* /* TITLE FIRST PAGE    */                                                                                    */     00000890
        PUT FILE(SYSPRINT) EDIT('SOURCE EDITED BY PLEDIT',  'PAGE ',PAGENO)                                       1 00000900
        (SKIP(2),COL(10),A,COL(NCOL-8),A,F(3)):                                                                   1 00000900
        PUT SKIP(2):                                                                                              1 00000930
/* /* INITIALIZE PUSHDOWN STACK    */                                                                           */     00000940
        ALLOCATE STACK1 INIT('TOP'),                                                                              1 00000950
             STACK2(5) INIT((5)''),                                                                               2 00000950
             STACK3 INIT(0):                                                                                      2 00000950
/* /* ADJUST LINELENGTH FOR LEVEL PRINTOUT    */                                                                */     00000980
        IF NCOL = 80 THEN                                                                                         1 00000990
             NCOL = 72:                                                                                           2 00000990
        ELSE                                                                                                      1 00001010
             NCOL = 116:                                                                                          2 00001010
/* /* PARAMETERS FOR CG SEARCH    */                                                                            */     00001020
        GET FILE(CCIN) LIST(BUFCG,TESTCG):                                                                        1 00001030
        HALFCG = BUFCG/TWOB:                                                                                      1 00001040
        IF TESTCG > HALFCG THEN                                                                                   1 00001050
                  DO:                                                                                             3 00001050
                  PUT SKIP LIST('TESCG MUST BE < HALF BUFCG'):                                                    3 00001070
                  STOP:                                                                                           3 00001090
                  END:                                                                                            3 00001100
/* /* GET STATEMENT SIZE  */                                                                                    */     00001101
        GET FILE(CCIN) LIST(STATEMENT_SIZE,MAX_LINES):                                                            1 00001102
        ALLOCATE TEXT(MAX_LINES),                                                                                 1 00001103
             PREFIX(MAX_LINES,5),                                                                                 2 00001103
             LABEL(MAX_LINES,5),                                                                                  2 00001103
             TYPE(MAX_LINES),                                                                                     2 00001103
             SEQ#(MAX_LINES),                                                                                       2 00001103
             LEVEL(MAX_LINES),                                                                                      2 00001103
             SKIP(MAX_LINES);                                                                                       2 00001103
        TEXT='';                                                                                                    1 00001107
        PREFIX='';                                                                                                  1 00001108
        LABEL='';                                                                                                   1 00001109
        TYPE='';                                                                                                    1 00001110
        LEVEL=0;                                                                                                    1 00001111
        SKIP=0;                                                                                                     1 00001112
        ALLOCATE TLINE CHAR(STATEMENT_SIZE) INIT('');                                                               1 00001113
        ALLOCATE NCARD CHAR(STATEMENT_SIZE) INIT('');                                                               1 00001114
        ALLOCATE LINE CHAR(STATEMENT_SIZE) INIT('');                                                                1 00001115
        ALLOCATE TEMLINE CHAR(STATEMENT_SIZE) INIT('');                                                             1 00001116
/* /* CREATE MARGIN TABLE    */                                                                                 */    00001117
        CALL SMARG;                                                                                                 1 00001120
/* ................................................................................................................ */
/* /* PART 11     */                                                                                            */    00001130
/* /* HERE WE START EDITING, 1 STATEMENT AT A TIME    */                                                        */    00001140
/* /* UPDATE POINTERS AND INDEX    */                                                                           */    00001150
GO:     COUNT = COUNT+ONEB;                                                                                         1 00001160
        RINDEX = ADDPTR(RINDEX);                                                                                    1 00001170
        LINDEX(COUNT) = RINDEX;                                                                                     1 00001180
/* /* GET 1ST. STORE CONDITION PRFIX AND/OR LABEL    */                                                         */    00001190
/* /* IF EOF, EMPTY BUFFER AND END    */                                                                        */    00001200
        CALL READ;                                                                                                  1 00001210
/* ................................................................................................................ */
/* /* COME HERE ON EOF    */                                                                                    */    00001220
ENDRD:  IF (FINIS) THEN                                                                                             1 00001230
             IF (¬RFLAG) THEN                                                                                       2 00001230
                       DO;                                                                                          4 00001230
                       COUNT = COUNT-ONEB;                                                                          4 00001260
                       RINDEX = SUBPTR(RINDEX);                                                                     4 00001270
                       GO TO IFCG;                                                                                  4 00001280
/* ................................................................................................................ */
                       END;                                                                                         4 00001290
             ELSE                                                                                                   2 00001300
                       DO;                                                                                          4 00001300
                       PUT SKIP LIST('**MISSING CARD(S)**');                                                        4 00001320
                       GO TO ENDPROG;                                                                               4 00001330
/* ................................................................................................................ */
                       END;                                                                                         4 00001340
/* /* NOT EOF, ANALYZE STATEMENT FOR TYPE,  GO TO TYPE ROUTINES, UPDATE  PUSHDOWN LIST, ASSIGN LEVEL AND       */    00001350
/* SKIPCODE, STORE IN BUFFER    */                                                                              */    00001350
/* /* IS IT COMMENT ?    */                                                                                     */    00001390
        IF COMFLG = ONEB THEN                                                                                       1 00001400
                  DO;                                                                                               3 00001400
                  CALL COMMENT;                                                                                     3 00001420
/* ................................................................................................................ */
                  GO TO P3;                                                                                         3 00001430
/* ................................................................................................................ */
                  END;                                                                                              3 00001440
/* /* NO, UPDATE STACKS    */                                                                                   */    00001450
        CALL PUSHPUL;                                                                                               1 00001460
/* ................................................................................................................ */
/* /* NULL ST. ?    */                                                                                          */    00001470
        IF LENGTH(LINE) = ONEB THEN                                                                                 1 00001480
             GO TO C8;                                                                                              2 00001480
/* ................................................................................................................ */
/* /* THFL IS CLUE TO TYPE    */                                                                                */    00001500
/* /* COMPLETED 'ELSE' ?   */                                                              */     00001510
        IF THFL = TWOB THEN                                                                     1 00001520
             GO TO P3:                                                                          2 00001520
/* ..................................................................................... */
/* /* FOR THFL = 0 OR 3   */                                                                */     00001570
/* /* IS IT PREPROCESSOR STATEMENT */                                                       */     00001571
        IF SUBSTR(LINE,1,1)='%' THEN                                                            1 00001572
                  DO:                                                                           3 00001572
                  COUNT=COUNT-1:                                                                3 00001573
                  GO TO TESTEN:                                                                 3 00001574
/* ..................................................................................... */
                  END:                                                                          3 00001575
/* /* IS IT AN 'IF' ?   */                                                                  */     00001580
C1:     CMND = SUBSTR(LINE,1,3);                                                                1 00001590
        IF (CMND = 'IF ')|(CMND = 'IF(')|  (CMND = 'IF''') THEN                                 1 00001600
             CALL IF:                                                                           2 00001600
/* ..................................................................................... */
/* /* IS IT DO, PROC, BEGIN OR ENTRY ?   */                                                 */     00001630
C2:     CMND = SUBSTR(LINE,1,6):                                                                1 00001640
        IF(CMND = 'ENTRY ')|(CMND = 'BEGIN ')|  (CMND = 'ENTRY:')|(CMND = '3EGIN:') THEN        1 00001650
             GO TO YESBL:                                                                       2 00001650
/* ..................................................................................... */
        CMND = SUBSTR(LINE,1,3):                                                                1 00001680
        IF (CMND = 'DO:')|(CMND = 'DO ') THEN                                                   1 00001690
             GO TO YESBL:                                                                       2 00001690
/* ..................................................................................... */
        CMND = SUBSTR(LINE,1,5):                                                                1 00001710
        IF (CMND = 'PROC:')|(CMND = 'PROC ')|  (CMND = 'PROC(') THEN                            1 00001720
             GO TO YESBL:                                                                       2 00001720
/* ..................................................................................... */
        CMND = SUBSTR(LINE,1,10):                                                               1 00001750
        IF (CMND ¬= 'PROCEDURE ') & (CMND ¬= 'PROCEDURE;')  & (CMND ¬= 'PROCEDURE(') THEN       1 00001760
             GO TO CC2:                                                                         2 00001760
/* ..................................................................................... */
/* /* IT IS A BLOCK COMMAND   */                                                            */     00001790
YESBL:  CALL BLOCK(CMND):                                                                       1 00001800
/* ..................................................................................... */
        GO TO P3:                                                                               1 00001810
/* ..................................................................................... */
/* /* TRY FOR 'END'   */                                                                    */     00001820
CC2:    CMND = SUBSTR(LINE,1,4):                                                                1 00001830
        IF(CMND = 'END ')|(CMND = 'END:') THEN                                                  1 00001840
                  DO:                                                                           3 00001840
                  CALL PEND:                                                                    3 00001860
/* ..................................................................................... */
                  GO TO P3:                                                                     3 00001870
/* ..................................................................................... */
                  END:                                                                          3 00001880
/* /* IS IT A CONTROL STATEMENT ?   */                                                      */     00001890
C4:     CMND = SUBSTR(LINE,1,5):                                                                1 00001900
        IF(CMND = 'CALL ') THEN                                                                 1 00001910
                  DO:                                                                           3 00001910
                  CALL CTRLC:                                                                   3 00001930
/* ..................................................................................... */
                  GO TO P3:                                                                     3 00001940
/* ..................................................................................... */
                  END:                                                                          3 00001950
        IF(CMND = 'GOTO ') THEN                                                                 1 00001960
             GO TO C41:                                                                         2 00001960
```

```
          IF(CMND = 'WAIT ')|(CMND = 'WAIT:')|  (CMND = 'STOP ')|(CMND = 'STOP:')|  (CMND = 'EXIT ')|(CMND =                    1 00001980
          'EXIT:') THEN                                                                                                         1 00001980
                GO TO C44:                                                                                                      2 00001980
/* ............................................................................................................................. */
          CMND = SUBSTR(LINE,1,6);                                                                                              1 00002020
          IF(CMND = 'DELAY ')|(CMND = 'DELAY:') THEN                                                                            1 00002030
                GO TO C44:                                                                                                      2 00002030
          IF (CMND = 'GO TO ') THEN                                                                                             1 00002050
C41:                    DO:                                                                                                     3 00002050
                        CALL CTRLG:                                                                                             3 00002070
/* ............................................................................................................................. */
                        GO TO P3:                                                                                               3 00002080
/* ............................................................................................................................. */
                        END:                                                                                                    3 00002090
          CMND = SUBSTR(LINE,1,7);                                                                                              1 00002100
          IF (CMND = 'RETURN ')|(CMND = 'RETURN:') THEN                                                                         1 00002110
C44:                    DO:                                                                                                     3 00002110
                        CALL CTRLO:                                                                                             3 00002130
/* ............................................................................................................................. */
                        GO TO P3:                                                                                               3 00002140
/* ............................................................................................................................. */
                        END:                                                                                                    3 00002150
/* /* NOT CONTROL ST    */                                                                                                */      00002160
/* /* IS IT IO ?    */                                                                                                   */      00002170
C5:       CMND = SUBSTR(LINE,1,4);                                                                                              1 00002180
          IF(CMND = 'GET ')|(CMND = 'PUT ') THEN                                                                                1 00002190
                GO TO C55:                                                                                                      2 00002190
/* ............................................................................................................................. */
          CMND = SUBSTR(LINE,1,5);                                                                                              1 00002210
          IF(CMND = 'OPEN ')|(CMND = 'READ ') THEN                                                                              1 00002220
                GO TO C55:                                                                                                      2 00002220
/* ............................................................................................................................. */
          CMND = SUBSTR(LINE,1,6);                                                                                              1 00002240
          IF(CMND = 'CLOSE ')|(CMND = 'WRITE ') THEN                                                                            1 00002250
                GO TO C55:                                                                                                      2 00002250
/* ............................................................................................................................. */
          CMND = SUBSTR(LINE,1,7);                                                                                              1 00002270
          IF(CMND = 'DELETE ')|(CMND = 'LOCATE ')|  (CMND = 'FORMAT ')|(CMND = 'FORMAT(')  THEN                                 1 00002280
                GO TO C55:                                                                                                      2 00002280
/* ............................................................................................................................. */
          CMND = SUBSTR(LINE,1,8);                                                                                              1 00002320
          IF(CMND = 'REWRITE ')|(CMND = 'DISPLAY ')  THEN                                                                       1 00002330
C55:                    DO:                                                                                                     3 00002330
                        CALL IOSUB:                                                                                             3 00002360
/* ............................................................................................................................. */
                        GO TO P3:                                                                                               3 00002370
/* ............................................................................................................................. */
                        END:                                                                                                    3 00002380
/* /* IS IT DECLARATICN ?    */                                                                                          */      00002390
C6:       IF(SUBSTR(LINE,1,4) = 'DCL ')|  (SUBSTR(LINE,1,8) = 'DEFAULT ')|  (SUBSTR(LINE,1,8) = 'DECLARE ')|                   1 00002400
          (SUBSTR(LINE,1,9) = 'ALLOCATE ') THEN                                                                                 1 00002400
                        DO:                                                                                                     3 00002400
                        CALL DECLARE:                                                                                           3 00002450
/* ............................................................................................................................. */
                        GO TO P3:                                                                                               3 00002460
/* ............................................................................................................................. */
                        END:                                                                                                    3 00002470
          IF(SUBSTR(LINE,1,5) = 'FREE ') THEN                                                                                   1 00002480
                GO TO C8:                                                                                                       2 00002480
```

```
/* ................................................................................................ */
/* /* IS IT ON, SIGNAL, REVERT ?   */                                                               */      00002500
C7:      IF(SUBSTR(LINE,1,3) = 'ON ')|  (SUBSTR(LINE,1,7) = 'SIGNAL ')|  (SUBSTR(LINE,1,7) = 'REVERT ') THEN   1 00002510
                   DO:                                                                                      3 00002510
                   CALL ONSUB:                                                                              3 00002550
/* ................................................................................................ */
                   GO TO P3:                                                                                3 00002560
/* ................................................................................................ */
                   END:                                                                                     3 00002570
/* /* NONE OF PRECEDING, CLASS IT ASSIGNMENT   */                                                    */      00002580
C8:      CALL ASSIGN:                                                                                       1 00002590
/* ................................................................................................ */
/* /* PART III    */                                                                                  */      00002600
/* /* RETURNED FROM TYPE SUBS, NOW CHECK  FOR OUTPUT, CG, ETC. TETURN TO 'GO'  IF INPUT REMAINS, OR TO END   */   00002610
/* PROGRAM  IF FINISHED.   */                                                                         */      00002610
P3:      THFL = ZB:                                                                                         1 00002650
/* /* POINTER TO FIRST BUFFER LINE OF  CURRENT STATEMENT   */                                         */      00002660
         TFISN = LINDEX(COUNT):                                                                             1 00002680
         NV1 = SKIP(THISN):                                                                                 1 00002690
/* /* IF THIS IS VERY FIRST ST, DC ANY  PRE-SKIP   */                                                 */      00002700
         IF(COUNT > 1) THEN                                                                                 1 00002720
              GO TC MORN1:                                                                                  2 00002720
/* ................................................................................................ */
         NV2 = CHCODE(NV1,ONEB):                                                                            1 00002740
         IF NV2 = ZB THEN                                                                                   1 00002750
              GO TO TESTFN:                                                                                 2 00002750
         IF(NV2 = ONEB)|(NV2=6)|(NV2 = 8) THEN                                                              1 00002770
              CALL OUTFST(NV2):                                                                             2 00002770
/* ................................................................................................ */
/* /* SUBTRACT PRE-SKIP FROM CODE   */                                                                */      00002790
         SKIP(THISN) = CHCODE(NV1,TWOB):                                                                    1 00002800
         GO TO TESTFN:                                                                                      1 00002810
/* ................................................................................................ */
/* /* NOT LINE 1: PTR TO LAST LINE FO PRECEDING ST   */                                               */      00002820
MORN1:   LASTN = SUBPTR(THISN):                                                                             1 00002830
         NV2 = SKIP(LASTN):                                                                                 1 00002840
/* /* RULE1 : FLAG OUTPUT IF CHANGE IN LEVEL   */                                                     */      00002850
RULE1:   IF(LEVEL(THISN) = LEVEL(LASTN)) THEN                                                               1 00002860
              LEVF = '0'B:                                                                                  2 00002860
         ELSE                                                                                               1 00002880
              LEVF = '1'B:                                                                                  2 00002880
/* /* RULE2 : EXAMINE CURRENT ST. FOR PRESKIP.  IF FOUND, ADD TO PREV. ST, SUBTRACT HERE   */           */      00002900
RULE2:   IF(NV1 = ONEB)|(NV1 = TWOB) THEN                                                                   1 00002920
              IF(NV2 = 4)|(NV2 = 5) THEN                                                                    2 00002920
                   SKIP(LASTN) = 5:                                                                         3 00002920
              ELSE                                                                                          2 00002950
                   SKIP(LASTN) = 3:                                                                         3 00002950
         ELSE                                                                                               1 00002970
              IF(NV1 = 6)|(NV1 = 7) THEN                                                                    2 00002970
                   SKIP(LASTN) = 5:                                                                         3 00002970
              ELSE                                                                                          2 00003000
                   IF(NV1 = 8)|(NV1 = 9) THEN                                                               3 00003000
                        IF(NV2 = ZB) THEN                                                                   4 00003000
                             SKIP(LASTN) = 4:                                                               5 00003000
                        ELSE                                                                                4 00003040
                             IF(NV2 = 3) THEN                                                               5 00003040
                                  SKIP(LASTN) = 5:                                                          6 00003040
         SKIP(THISN) = CHCODE(SKIP(THISN),TWOB):                                                            1 00003070
/* /* RULE3 D IMPLEMENT SKIPS FOR PRECEDING ST   */                                                   */      00003080
/* /* ALSO OUTPUT IF LEVEL CHANGE   (RULE1)   */                                                                    */    00003090
RULE3:  NV2 = SKIP(LASTN);                                                                                              1 00003110
        IF(LEVF)|(NV2 ¬= ZB) THEN                                                                                       1 00003120
                  DO;                                                                                                   3 00003120
                  CALL OUTPUT(COUNT-ONEB);                                                                              3 00003140
                  CALL MOVUP(COUNT-ONEB);                                                                               3 00003150
/* ................................................................................................................. */
                  GO TO TESTFN;                                                                                         3 00003160
/* ................................................................................................................. */
                  END;                                                                                                  3 00003170
/* /* NO OUTPUT YET. LOOK FOR CGIF   BUFFER = BUFCG OR IF (EOF AND   BUFFER > HALFCG).   */                         */    00003180
IFCG:   IF (COUNT = BUFCG) | ((COUNT >= HALFCG)  & FINIS) THEN                                                          1 00003210
             GO TO RULE4;                                                                                               2 00003210
/* ................................................................................................................. */
/* /* NOT FULL ENOUGH, READ MORE UNLESS   NO MORE INPUT   */                                                         */    00003240
TESTFN: IF(¬FINIS) THEN                                                                                                 1 00003260
             GO TO GO;                                                                                                  2 00003260
/* ................................................................................................................. */
/* /* EMPTY BUFFER AT END    */                                                                                      */    00003280
OUTEND: CALL OUTPUT(COUNT);                                                                                             1 00003290
/* ................................................................................................................. */
        GO TO ENDPROG;                                                                                                  1 00003300
/* ................................................................................................................. */
/* /* SEARCH FOR CG   */                                                                                             */    00003310
RULE4:  IF(FINIS) THEN                                                                                                  1 00003320
             CALL CGEND(NST,NEND);                                                                                      2 00003320
        ELSE                                                                                                            1 00003340
             CALL CGFIND(NST,NEND);                                                                                     2 00003340
/* ................................................................................................................. */
/* /* NOT FOUND, OUTPUT HALF BUFFER   (OR ALL, IF END OF DATA)   */                                                  */    00003360
        IF(NST = ZB) THEN                                                                                               1 00003380
             IF(FINIS) THEN                                                                                             2 00003380
                  GO TO OUTEND;                                                                                         3 00003380
             ELSE                                                                                                       2 00003410
                       DO;                                                                                              4 00003410
                       CALL OUTPUT(HALFCG);                                                                             4 00003430
                       CALL MOVUP(HALFCG);                                                                              4 00003440
/* ................................................................................................................. */
                       GO TO GO;                                                                                        4 00003450
/* ................................................................................................................. */
                       END;                                                                                             4 00003460
/* /* FOUNDCG, OUTPUT PRE-CG LINES    */                                                                             */    00003470
        IF(NST = ONEB) THEN                                                                                             1 00003480
             GO TO CGOUT;                                                                                               2 00003480
/* ................................................................................................................. */
        PT = SUBPTR(LINDEX(NST));                                                                                       1 00003500
        SKIP(PT) = 3;                                                                                                   1 00003510
        NST = NST-ONEB;                                                                                                 1 00003520
        CALL OUTPUT(NST);                                                                                               1 00003530
        CALL MOVUP(NST);                                                                                                1 00003540
/* ................................................................................................................. */
/* /* OUTPUT CG   */                                                                                                 */    00003550
        NEND = NEND-NST;                                                                                                1 00003560
CGOUT:  IF(NEND = COUNT) THEN                                                                                           1 00003570
             GO TO OUTGP;                                                                                               2 00003570
/* ................................................................................................................. */
        PT = SUBPTR(LINDEX(NEND+ONEB));                                                                                 1 00003590
        SKIP(PT) = 3;                                                                                                   1 00003600
OUTGP:  CALL OUTPUT(NEND);                                                                                              1 00003610
```

```
        CALL MOVUP(KEND);                                                                                            1 00003620
/* ............................................................................................................... */
        GO TO TESTFN;                                                                                                1 00003630
/* ............................................................................................................... */
/* /* ALL INPUT PROCESSED, ALL OUTPUT  DONE, TELL IT.    */                                                      */    00003640
ENDPROG:PUT PAGE LIST ('**PLEDIT FINISHED**');                                                                       1 00003660
/* /* ******** PAGE 20 ON HANDWRITTEN SHEETS ******** */                                                         */    00003680
/* /* SUBROUTINE  READ GETS INPUT AND  BEGINS TO PROCESS STATEMENT  */                                           */    00003710
READ:        PROC;                                                                                                   2 00003730
/* /*  GETSTAT PUTS STATEMENT INTO LINE    */                                                                    */    00003740
             CALL  GETSTAT;                                                                                          2 00003750
/* ............................................................................................................... */
/* /* COMMENTS ARE NOT PARSED */                                                                                 */    00003760
             IF (COMFLG = ONEB) THEN                                                                                 2 00003770
                  RETURN;                                                                                            3 00003770
/* /* RLABEL SEPARATES LABEL(S) AND  CONDITION PREFIX(ES)  */                                                    */    00003790
             CALL RLABEL;                                                                                            2 00003810
/* ............................................................................................................... */
/* /* NOW RETURN IS MADE TO PLEDIT   */                                                                          */    00003820
             END READ;                                                                                               2 00003830
GETSTAT:     PROC;                                                                                                   2 00003840
             DCL CLABL(0:2) LABEL INIT(ISITC,COMT,NOCOMT),                                                           2 00003850
                  (NQUOT,NCOM) BIT(1) INIT('0'B),                                                                    3 00003850
                  KCH CHAR(1),                                                                                       3 00003850
                  KCC CHAR(2),                                                                                       3 00003850
                  KS BIN FIXED(15,0);                                                                                3 00003850
             COMFLG = ZB;                                                                                            2 00003900
/* /* IF MORE TEXT NEEDED, READ NEW RECORD   */                                                                  */    00003910
             IF (NCARD = '') THEN                                                                                    2 00003920
G1:                    DO;                                                                                           4 00003920
                       RFLAG = '0'B;                                                                                 4 00003940
                       CALL IN;                                                                                      4 00003950
/* ............................................................................................................... */
                       END G1;                                                                                       4 00003960
/* /* SUBROUTINE IN READS SYSIN (SYSTEM FILE)   */                                                               */    00003970
IN:               PRCC;                                                                                              3 00003980
/* /* READ CARD COL2 - 72, APPEND TO NCARD   */                                                                  */    00003990
/* /* ON EOF, FINIS = 1 AND RETURN MADE TO ENDRD (IN PLEDIT)*/                                                   */    00004000
AGIN:             GET FILE(SYSIN) EDIT(CONT_CHAR,LINE)  (A(1),A(79));                                                3 00004010
                  IF NCARD='' THEN                                                                                   3 00004013
                       SEQNO=SUBSTR(LINE,72);                                                                        4 00004013
                  LINE = SUBSTR(LINE,1,71);                                                                          3 00004030
                  IF CONT_CHAR='$' THEN                                                                              3 00004031
                            DO;                                                                                      5 00004031
                            CALL OUTPUT(COUNT-ONEB);                                                                 5 00004032
                            CALL MOVUP(COUNT-ONEB);                                                                  5 00004033
/* ............................................................................................................... */
                            PUT EDIT(LINE)(SKIP(1),X(1),A);                                                          5 00004034
                            GET FILE(SYSIN) EDIT(CONT_CHAR,LINE)  (A(1),A(79));                                      5 00004035
                            LINE=SUBSTR(LINE,1,71);                                                                  5 00004037
                                 DO WHILE(CONT_CHAR¬='$');                                                           6 00004038
                                 PUT EDIT(LINE)(SKIP(1),X(1),A);                                                     6 00004039
                                 GET FILE(SYSIN) EDIT(CONT_CHAR,LINE)  (A(1),A(79));                                 6 00004040
                                 LINE=SUBSTR(LINE,1,71);                                                             6 00004042
                                 END;                                                                                6 00004043
                            PUT EDIT(LINE)(SKIP(1),X(1),A);                                                          5 00004044
                            GO TO AGIN;                                                                              5 00004045
/* ............................................................................................................... */
                            END;                                                                                     5 00004046
```

```
                 IF LINE=' ' THEN                                                                                3 00004047
                      GO TO AGIN;                                                                                4 00004047
/* ............................................................................................................ */
                      DO I=71 TO 1 BY -1 WHILE(SUBSTR(LINE,I,1)=' ');                                            4 00004048
                      END;                                                                                       4 00004049
                 IF I<71 THEN                                                                                    3 00004050
                      I=I+1;                                                                                     4 00004050
                 LINE=SUBSTR(LINE,1,I);                                                                          3 00004051
                      DO I=1 TO 71 WHILE(SUBSTR(LINE,I,1)=' ');                                                  4 00004052
                      END;                                                                                       4 00004053
                 IF I>1 THEN                                                                                     3 00004054
                      I=I-1;                                                                                     4 00004054
                 LINE=SUBSTR(LINE,I);                                                                            3 00004055
                 I=LENGTH(NCARD)+LENGTH(LINE);                                                                   3 00004056
                 IF I>STATEMENT_SIZE THEN                                                                        3 00004057
                           DO;                                                                                   5 00004057
                           PUT EDIT('STATEMENT SIZE EXCEEDED.  ')(PAGE,A);                                       5 00004058
                           PUT EDIT(' NCARD: ',NCARD)(SKIP(1),2 A);                                              5 00004059
                           PUT EDIT('  LINE: ',LINE)(SKIP(1),2 A);                                               5 00004060
                           STOP;                                                                                 5 00004061
                           END;                                                                                  5 00004062
                 NCARD = NCARD || LINE;                                                                          3 00004063
                 END IN;                                                                                         3 00004064
/* /* BRANCH FOR TEST(COMFLG = 0), COMMENT(1), NO(2)   */                                                    */    00004065
C1:         GO TO CLABL(COMFLG);                                                                                 2 00004070
/* ............................................................................................................ */
/* /* DELETE LEADING BLANKS   */                                                                            */    00004080
/* /* IS IT COMMENT ?   */                                                                                  */    00004090
ISITC:      NCARD = SUBSTR(NCARD,VERIFY(NCARD,BLANK));                                                           2 00004100
            IF INDEX(NCARD,BRK(6)) = 1 THEN                                                                      2 00004110
                      DO;                                                                                        4 00004110
                      COMFLG = ONEB;                                                                             4 00004130
                      GO TO COMT;                                                                                4 00004140
/* ............................................................................................................ */
                      END;                                                                                       4 00004150
/* /* STATEMENT IS NOT COMMENT, WILL END IN  SEMICOLON (NOT IN QUOTES OR  COMMENT)   */                        */    00004160
            COMFLG = TWOB;                                                                                       2 00004190
NOCOMT:          DO KS = 1 TO LENGTH(NCARD);                                                                     3 00004200
                 KCH=SUBSTR(NCARD,KS,1);                                                                         3 00004210
                 IF(KCH = BRK(3)) & (¬NQUOT) & (¬NCOM)  THEN                                                     3 00004220
                      GO TO FBRK;                                                                                4 00004220
                 IF (KCH = BRK(1)) THEN                                                                          3 00004250
                      NQUOT = ¬NQUOT;                                                                            4 00004250
                 ELSE                                                                                            3 00004270
                      IF KS¬=LENGTH(NCARD) THEN                                                                  4 00004270
                           IF (SUBSTR(NCARD,KS,2)=BRK(6) | SUBSTR(NCARD,KS,2)=BRK(7)) THEN                       5 00004270
                                NCOM = ¬NCOM;                                                                    6 00004270
                 END NOCOMT;                                                                                     3 00004310
/* /* NO ENDBREAK FOUND. GET MORE TEXT   */                                                                 */    00004320
MORE:       RFLAG = '1'B;                                                                                        2 00004330
            CALL IN;                                                                                             2 00004340
/* ............................................................................................................ */
            GO TO C1;                                                                                            2 00004350
/* ............................................................................................................ */
/* /* FOUND ENDBREAK. STORE ST. IN LINE   */                                                                */    00004360
FBRK:       LINE = SUBSTR(NCARD,1,KS);                                                                           2 00004370
/* /* CLEAR ST. FORM NCARD   */                                                                             */    00004380
            NCARD = SUBSTR(NCARD,KS+1);                                                                          2 00004390
            RETURN;                                                                                              2 00004400
```

```
/* /* COMMENT ST.. FIND END    */                                                   */   00004410
COMT:            DO KS = 3 TO LENGTH(NCARD) - 1:                                       3 00004420
                 KCC = SUBSTR(NCARD,KS,2):                                             3 00004430
                 IF (KCC = BRK(7)) THEN                                                3 00004440
                           DO:                                                         5 00004440
                           KS = KS + 1:                                                5 00004460
                           GO TC FBRK:                                                 5 00004470
/* .............................................................................................*/
                 END COMT:                                                             3 00004480
/* /* NO END BREAK FCUND   */                                                       */   00004490
            GO TC MCRE:                                                                2 00004500
/* .............................................................................................*/
            END GETSTAT:                                                               2 00004510
/* /* RLABEL IS CALLED BY PLEDIT, IF AND  ELSE TO SEPARATE ALL PREFIXES   */        */   00004520
RLABEL:     PROC:                                                                      2 00004540
            DCL (KLA,NLAB INIT (0)) BIN FIXED(15,0),                                   2 00004550
                 KF    CHAR:                                                           3 00004550
/* /* UP TO 31 CHAR IN A PREFIX   */                                                */   00004570
/* /* FIND ANY CONDITION PREFIX(ES)   */                                            */   00004580
            CALL SPREFX:                                                               2 00004590
/* .............................................................................................*/
SPREFX:          PROC:                                                                 3 00004600
                 DCL (IP,NP INIT(0)) BIN FIXED (15,0):                                 3 00004610
/* /* SET PREFIX ARRAY TO NULL   */                                                 */   00004620
                 PREFIX(RINDEX,*) = MTLAB:                                             3 00004630
ISITP:           NP = NP+ONEB:                                                         3 00004640
                 LINE = SUBSTR(LINE,VERIFY(LINE,BLANK)):                               3 00004650
                 IF SUBSTR(LINE,1,1)¬=BRK(4) THEN                                      3 00004660
                      RETURN:                                                          4 00004660
/* /* FOUND PREFIX START, LOOK FOR END   */                                         */   00004680
LPP:                  DO IP = 2 TO LENGTH(LINE) - 1:                                   4 00004690
                      IF SUBSTR(LINE,IP,2)   ¬=')':' THEN                              4 00004700
                           GO TO DEND:                                                 5 00004700
/* .............................................................................................*/
                      PREFIX(RINDEX,NP) = SUBSTR(LINE,1,IP + 1):                       4 00004720
                      LINE = SUBSTR(LINE,IP + 2):                                      4 00004730
/* /* REPEAT FOR ANOTHER   */                                                       */   00004740
                      IF NP < 5 THEN                                                   4 00004750
                           GO TO ISITP:                                                5 00004750
                      ELSE                                                             4 00004770
                           RETURN:                                                     5 00004770
DEND:                 END LPP:                                                         4 00004780
/* /* NO END, ERROR.  TRY TO GO ON   */                                             */   00004790
                 PUT SKIP LIST('**UNBALANCED PARENS IN PREFIX**'):                     3 00004800
                 IP = INDEX(LINE,BRK(8)):                                              3 00004820
                 IF IP = 0 THEN                                                        3 00004830
                      STOP:                                                            4 00004830
                 PREFIX(RINDEX,NP) = SUBSTR(LINE,1,IP):                                3 00004850
                 LINE = SUBSTR(LINE,IP + 1):                                           3 00004860
                 IF NP < 5 THEN                                                        3 00004870
                      GO TO ISITP:                                                     4 00004870
/* .............................................................................................*/
                 END SPREFX:                                                           3 00004890
/* /* LCOK FOR LABEL(S).  COLON MUST COME  BEFORE BLANK, QUCTE OR LEFT PAREN   */   */   00004900
            LABEL(RINDEX,*) = MTLAB:                                                   2 00004920
/* /* ALWAYS DELETE LEADING BLANKS   */                                             */   00004930
RL1:        LINE = SUBSTR(LINE,VERIFY(LINE,BLANK)):                                    2 00004940
            NLAB = NLAB + 1:                                                           2 00004950
RLP:             DO KLA = 1 TO LENGTH(LINE) - 1:                                       3 00004960
```

```
                  KH = SUBSTR(LINE,KLA,1);                                                                          3 00004970
                  IF (KH = BRK(1)) | (KH = BLANK) | (KH = BRK(4))  THEN                                             3 00004980
                       RETURN;                                                                                      4 00004980
                  IF KH = BRK(8) THEN                                                                               3 00005010
                            DO;                                                                                     5 00005010
                            LABEL(RINDEX,NLAB) =  SUBSTR(LINE,1,KLA);                                               5 00005030
                            LINE = SUBSTR(LINE,KLA+1);                                                              5 00005050
                            IF NLAB < 5 THEN                                                                        5 00005060
                                 GO TO RL1;                                                                         6 00005060
/* ............................................................................................................. */
                            END;                                                                                    5 00005080
                  END RLP;                                                                                          3 00005090
             END RLABEL;                                                                                            2 00005100
/* /* FUNCTION ADDPTR AND SUBPTR ARE CALLED  TO FIND POINTER TO NEXT (PRECEDING) LINE  IN BUFFER. BUFFER SIZE    */   00005110
/* IS SET TO 50   */                                                                                             */   00005110
ADDPTR:      PROC (PT) RETURNS (BIN FIXED(15,0));                                                                   2 00005140
/* /* INCREMENT BUFFER POINTER    */                                                                             */   00005150
             DCL(PNEXT,PT) BIN FIXED(15,0);                                                                         2 00005160
             PNEXT = PT + ONEB;                                                                                     2 00005170
             IF PNEXT > MAX_LINES THEN                                                                              2 00005180
                  PNEXT = ONEB;                                                                                     3 00005180
             RETURN (PNEXT);                                                                                        2 00005260
             END ADDPTR;                                                                                            2 00005270
SUBPTR:      PROC(PT) RETURNS (BIN FIXED(15,0));                                                                    2 00005280
/* /* FINDS POINTER TO PRECEDING BUFFER LINE    */                                                               */   00005290
             DCL PT BIN FIXED(15,0);                                                                                2 00005300
             IF (PT - ONEB)> ZB THEN                                                                                2 00005310
                  RETURN (PT - ONEB);                                                                               3 00005310
             ELSE                                                                                                   2 00005330
                  RETURN(MAX_LINES);                                                                                3 00005330
             END SUBPTR;                                                                                            2 00005350
/* /* ASSIGNMENT STATEMENTS, FREE ST., ETC.    */                                                                */   00005360
ASSIGN:      PROC;                                                                                                  2 00005370
/* /* ARRIVES HERE BY FALLING THROUGH ALL OTHER  CLASSIFICATION TESTS OR AS 'FREE' ST.   */                      */   00005380
             IF(LENGTH(LINE)<6)  | (SUBSTR(LINE,1,5)   ¬= 'FREE ')  THEN                                            2 00005400
                  TYPE(RINDEX) = LTYPE(1);                                                                          3 00005400
             ELSE                                                                                                   2 00005430
                  TYPE(RINDEX) = LTYPE(5);                                                                          3 00005430
             SKIP(RINDEX) = 0;                                                                                      2 00005450
             LEVEL(RINDEX) = CLEVEL;                                                                                2 00005460
             CALL STEXT(LINE);                                                                                      2 00005470
/* ............................................................................................................. */
             END ASSIGN;                                                                                            2 00005480
/* /* PROCEDURE FOR BEGIN,DO, ENTRY,  PROC STATEMENTS   */                                                       */   00005490
BLOCK:       PROC (CMND);                                                                                           2 00005510
             DCL CMND CHAR(*) VARYING;                                                                              2 00005520
/* /* ENTRY SKIPS LINE, NO LEVEL CHANGE    */                                                                    */   00005530
             IF (CMND = 'ENTRY') THEN                                                                               2 00005540
                       DO;                                                                                          4 00005540
                       SKIP(RINDEX) = 1;                                                                            4 00005560
                       FLAG = '0'B;                                                                                 4 00005570
                       END;                                                                                         4 00005580
             ELSE                                                                                                   2 00005590
                       DO;                                                                                          4 00005590
                       SKIP(RINDEX) = 0;                                                                            4 00005610
                       FLAG = '1'B;                                                                                 4 00005620
                       END;                                                                                         4 00005630
/* /* ENTER IN STACK WITH LABEL(S)    */                                                                         */   00005640
             CALL PUSHDON (CMND,LABEL(RINDEX,*));                                                                   2 00005650
```

```
 /* .............................................................................................. */
             TYPE(RINDEX) = LTYPE(4);                                                          2 00005660
             LEVEL(RINDEX) = CLEVEL;                                                           2 00005670
             CALL STEXT(LINE);                                                                 2 00005680
 /* .............................................................................................. */
             END BLOCK;                                                                        2 00005690
 /* /* THIS ROUTINE STORES COMMENT STATEMENTS    */                                        */   00005700
 COMMENT:    PROC;                                                                             2 00005710
             TYPE(RINDEX) = LTYPE(7);                                                          2 00005720
 /* /* ALL COMMENTS ARE LEVEL 0    */                                                       */   00005730
             LEVEL(RINDEX) = 0;                                                                2 00005740
             IF (COUNT = 1) THEN                                                               2 00005750
                  GO TO CSKIP;                                                                 3 00005750
 /* .............................................................................................. */
 /* /* GROUP COMMENTS, SPACE BEFORE FIRST  AND AFTER LAST    */                             */   00005760
             IF TYPE(LINDEX(COUNT-ONEB)) = LTYPE(7)  THEN                                      2 00005780
                       DO;                                                                     4 00005780
                       PT = SUBPTR(RINDEX);                                                    4 00005810
                       IF (SKIP(PT) = TWOB) THEN                                               4 00005820
                            SKIP (PT) = ONEB;                                                  5 00005820
                       ELSE                                                                    4 00005840
                            SKIP (PT) = ZB;                                                    5 00005840
                       SKIP(RINDEX) = 3;                                                       4 00005860
                       END;                                                                    4 00005870
             ELSE                                                                              2 00005880
 CSKIP:           SKIP(RINDEX) = TWOB;                                                         3 00005880
             CALL STEXT(LINE);                                                                 2 00005900
 /* .............................................................................................. */
             END COMMENT;                                                                      2 00005910
 /* /* CGFIND ( OR CGEND ) SEARCHES BUFFER FOR  CONCEPTUAL GROUPS   */                      */   00005920
 CGFIND:     PROC (NST,NEND);                                                                  2 00005940
             DCL  (LCT(3),LT,KC,JC,L2END,TEND,JEND)   BIN FIXED(15,0),                         2 00005950
                  STYPE CHAR(4) VAR,                                                           3 00005950
                  FTYPE CHAR(4) VAR INIT ('');                                                 3 00005950
 /* /* CONSIDER EACH GROUP OF HALFCG  STATEMENTS, STARTING AT TOP OF BUFFER  AND CONTINUING UNTIL BUFFER BOTTOM   */   00005990
 /* -1 IS HIT. IF TESTCG  STATEMENTS OF A GROUP ARE OF ONE  TYPE (ASSIGNMENT, IO, OR CALL),  RETURN NST = # OF   */   00005990
 /* FIRST CG ST.,  NEND = # OF LAST CG ST.    */                                            */   00005990
             JEND = HALFCG;                                                                    2 00006070
             TEND = COUNT - ONEB;                                                              2 00006080
 L0:         NST,NEND = ZB;                                                                    2 00006090
 L1:              DO  JC = ONEB TO JEND;                                                       3 00006100
                  LCT = ZB;                                                                    3 00006110
                  L2END = JC + HALFCG - ONEB;                                                  3 00006120
 L2:                   DO KC = JC TO L2END;                                                    4 00006130
                       STYPE = TYPE(LINDEX(KC));                                               4 00006140
 /* /* CONTERS FOR CALL, IO, ASSIGN ST.    */                                               */   00006150
                            DO LT = 1 TO 3;                                                    5 00006160
                            IF LTYPE(LT) = STYPE THEN                                          5 00006170
                                      DO;                                                      7 00006170
                                      LCT(LT) = LCT(LT) + ONEB;                                7 00006190
                                      IF LCT(LT) = TESTCG THEN                                 7 00006200
                                                DO;                                            9 00006200
                                                FTYPE = LTYPE(LT);                             9 00006220
                                                NEND = KC;                                     9 00006230
                                                GO TO L3;                                      9 00006240
 /* .............................................................................................. */
                                                END;                                           9 00006250
                       END L2;                                                                 4 00006260
 /* /* NO CG IF NEND IS STILL 0    */                                                       */   00006270
```

```
                  IF (NEND = ZB) THEN                                                                   3 00006280
                       GO TO L4;                                                                        4 00006280
/* ............................................................................................... */
/* /* FOUND CG. DOES IT EXTEND FURTHER ?   */                                                     */   00006300
L3:               IF (NEND=TEND) THEN                                                                   3 00006310
                       GO TO FIRST;                                                                     4 00006310
/* ............................................................................................... */
                       DO KC = NEND + ONEB TO TEND;                                                     4 00006330
                       IF TYPE (LINDEX(KC)) = FTYPE THEN                                                4 00006340
                            NEND = KC;                                                                  5 00006340
                       ELSE                                                                             4 00006360
                            GO TO FIRST;                                                                5 00006360
/* ............................................................................................... */
                       END;                                                                             4 00006380
/* /* FIND FIRST CG-TYPE STATEMENT   */                                                           */   00006390
FIRST:                 DO KC = JC TO JC + HALFCG - TESTCG;                                              4 00006400
                       IF TYPE(LINDEX(KC)) = FTYPE THEN                                                 4 00006410
                                 DO;                                                                    6 00006410
                                 NST = KC;                                                              6 00006430
                                 RETURN;                                                                6 00006440
                                 END;                                                                   6 00006450
L4:               END  L1;                                                                              3 00006460
/* /* NO CG. NST AND NEND STILL 0   */                                                            */   00006470
             RETURN;                                                                                    2 00006480
/* /* ENTER HERE TO SEARCH FOR CG IN  PARTIALLY FILLED BUFFER AT END   */                         */   00006490
CGEND:       ENTRY(NST,NEND);                                                                           2 00006510
             JEND = COUNT + ONEB - HALFCG;                                                              2 00006520
             TEND = COUNT;                                                                              2 00006530
             GO TO LC;                                                                                  2 00006540
/* ............................................................................................... */
             END CGFIND;                                                                                2 00006550
/* -1 IS HIT. IF TESTCG  STATEMENTS OF A GROUP ARE OF ONE  TYPE (ASSIGNMENT, IO, OR CALL), 'RETURN NST = # OF   */   00005990
/* /* ********* PAGE 36 IN HANDWRITTEN COPY ********* */                                           */   00006570
/* /* PROCESSES CALL, GO TO, EXIT, STOP, WAIT,  DELAY, RETURN STATEMENTS   */                      */   00006600
CTRLC:       PROC;                                                                                      2 00006620
/* /* ENTRY FOR CALL   */                                                                         */   00006630
             TYPE(RINDEX) = LTYPE(3);                                                                   2 00006640
             IF (COUNT = ONEB) THEN                                                                     2 00006650
                  GO TO ONC1;                                                                           3 00006650
/* ............................................................................................... */
/* /* FIND PRECEDING LINE SKIPCODE, NO  SKIP BETWEEN SUCCESSIVE CALLS   */                        */   00006660
             PT = SUBPTR(RINDEX);                                                                       2 00006680
             IF (SKIP(PT) = 5) & (TYPE(LINDEX(COUNT - ONEB))  = LTYPE(3)) THEN                          2 00006690
                  SKIP(PT) = ZB;                                                                        3 00006690
ONC1:        SKIP(RINDEX) = 5;                                                                          2 00006720
ONC2:        IF (PLEV¬=0) THEN                                                                          2 00006730
                       DO;                                                                              4 00006730
                       LEVEL(RINDEX) = PLEV;                                                            4 00006750
                       PLEV = 0;                                                                        4 00006760
                       END;                                                                             4 00006770
             ELSE                                                                                       2 00006780
                  LEVEL(RINDEX)=CLEVEL;                                                                 3 00006780
             CALL STEXT(LINE);                                                                          2 00006800
/* ............................................................................................... */
             RETURN;                                                                                    2 00006810
/* /* ENTRY FOR GO TO ST.   */                                                                    */   00006820
CTRLG:            ENTRY;                                                                                3 00006830
                  TYPE(RINDEX) = LTYPE(4);                                                              3 00006840
                  GO TO ONC1;                                                                           3 00006850
```

```
/* ..................................................................................................... */
/* /* ENTRY FOR ALL OTHER CONTROL ST.   */                                                              */     00006860
CTRLO:                ENTRY;                                                                                4 00006870
                      TYPE(RINDEX) = LTYPE(4);                                                              4 00006880
                      SKIP(RINDEX) = 3;                                                                     4 00006890
                      GO TO ONC2;                                                                           4 00006900
/* ..................................................................................................... */
            END CTRLC;                                                                                      2 00006910
DECLARE:    PROC;                                                                                           2 00006920
/* /* FOR DECLARATIONS, ALLOCATE AND DEFAULT ST.   */                                                   */     00006930
            DCL (NCHAR,L) BIN FIXED(15,0),                                                                  2 00006940
                 SLEV DEC FIXED(2);                                                                         3 00006940
/* /* FIRST LINE OF STATEMENT    */                                                                     */     00006960
            LEVEL(RINDEX) = CLEVEL;                                                                         2 00006970
            TYPE(RINDEX) = LTYPE(5);                                                                        2 00006980
            SKIP(RINDEX) = ONEB;                                                                            2 00006990
            NCHAR = 5;                                                                                      2 00007000
/* /* SUBST 'DCL' FOR FULL WORD   */                                                                    */     00007010
            IF (SUBSTR(LINE,1,7) = 'DECLARE') THEN                                                          2 00007020
                 LINE = 'DCL' || SUBSTR(LINE,8);                                                            3 00007020
/* /* SEPARATE PHASES : FIND FIRST COMMA  NOT IN QUOTE OR PARENS   */                                   */     00007040
FCOMMA:     CALL FINDCOM(L);                                                                                2 00007060
/* ..................................................................................................... */
            TEMLINE = SUBSTR(LINE,1,L);                                                                     2 00007070
/* /* FIRST LINE STARTS AT CURRENT MATGIN   */                                                          */     00007080
            IF (NCHAR = 5) THEN                                                                             2 00007090
                 GO TO PUT1;                                                                                3 00007090
/* ..................................................................................................... */
/* /* FOR OTHER LINES   */                                                                              */     00007110
/* /* FIND MARGIN, PREPARE TO STORE   */                                                                */     00007120
            IF SUBSTR(TEMLINE,1,2)=BRK(6) THEN                                                              2 00007121
                      DO;                                                                                   4 00007121
                      TLINE=LINE;                                                                           4 00007122
                      L=INDEX(TEMLINE,BRK(7))+1;                                                            4 00007123
                      LINE=SUBSTR(TEMLINE,1,L);                                                             4 00007124
                      CALL COMMENT;                                                                         4 00007125
/* ..................................................................................................... */
                      LINE=TLINE;                                                                           4 00007126
                      GO TO PUT1A;                                                                          4 00007127
/* ..................................................................................................... */
                      END;                                                                                  4 00007128
            IF (VERIFY(SUBSTR(TEMLINE,1,1),'0123456789')  = ZB) THEN                                        2 00007130
                 SLEV=DEC(SUBSTR(TEMLINE,1,INDEX(TEMLINE,' ')-1),2,0);                                      3 00007130
            ELSE                                                                                            2 00007160
                 SLEV = 1;                                                                                  3 00007160
            LEVEL(RINDEX) = CLEVEL + SLEV;                                                                  2 00007180
            SKIP(RINDEX) = ZB;                                                                              2 00007190
            PREFIX(RINDEX,*)='';                                                                            2 00007200
            TYPE(RINDEX)='';                                                                                2 00007201
            LABEL(RINDEX,*)='';                                                                             2 00007202
/* /* STORE PHASE IN BUFFER   */                                                                        */     00007220
PUT1:       CALL STEXT (TEMLINE);                                                                           2 00007230
/* ..................................................................................................... */
/* /* MOVE LINE UP OPERATE ON NEXT PART   */                                                            */     00007240
PUT1A:      LINE = SUBSTR(LINE,L + 1);                                                                      2 00007241
/* /* DELETE LEADING BLANKS   */                                                                        */     00007260
            IF VERIFY(LINE,BLANK)¬=0 THEN                                                                   2 00007270
                 LINE = SUBSTR(LINE,VERIFY(LINE,BLANK));                                                    3 00007270
/* /* FINISHED ?   */                                                                                   */     00007290
```

```
             IF LINE = '' THEN                                                    2 00007300
                  GO TO DCLEND;                                                   3 00007300
/* ............................................................................ */
/* /* NO, REPEAT    */                                                         */   00007320
             NCHAR = CNEB;                                                        2 00007330
             RINDEX = ADDPTR(RINDEX);                                             2 00007340
             GO TO FCCMMA;                                                        2 00007350
DCLEND:      IF SKIP(RINDEX) = ONEB THEN                                          2 00007360
                  SKIP(RINDEX) = TWOB;                                            3 00007360
             ELSE                                                                 2 00007380
                  SKIP(RINDEX) = 3;                                               3 00007380
             END    DECLARE;                                                      2 00007400
/* /* ELSE IS CALLED BY PUSHPUL    */                                          */   00007410
ELSE:        PROC;                                                                2 00007420
             TYPE(RINDEX) = LTYPE(4);                                             2 00007440
             SKIP(RINDEX) = ZB;                                                   2 00007450
/* /* POP UP USED UP IFS */                                                    */   00007451
                  DO WHILE(STACK1='ELSE');                                        3 00007452
                  CALL POPUP('ELSE');                                             3 00007453
/* ............................................................................ */
                  CLEVEL=STACK3-1;                                                3 00007454
                  CALL POPIF;                                                     3 00007455
/* ............................................................................ */
                  END;                                                            3 00007456
/* /* CHECK FOR MATCHING IF    */                                              */   00007460
/* /* GB   */                                                                  */   00007470
             IF STACK1¬='IF' THEN                                                 2 00007471
                       DO;                                                        4 00007471
                       PUT SKIP LIST('**ERROR IN IF...ELSE STRUCTURE**');         4 00007472
                       STOP;                                                      4 00007473
                       END;                                                       4 00007474
/* /* END OF GB  */                                                            */   00007475
/* /* ENTER IN PUSHDOWN LIST WITH NULL LABEL    */                             */   00007490
             FLAG='0'B;                                                           2 00007500
             CALL PUSHDON('ELSE',MTLAB);                                          2 00007501
/* ............................................................................ */
             CLEVEL=STACK3;                                                       2 00007502
             LEVEL(RINDEX)=CLEVEL-1;                                              2 00007503
/* /* IS ELSE FOLLOWED BY SEMICOLON (EMPTY)    */                              */   00007520
             TEMLINE = SUBSTR(LINE,5);                                            2 00007530
             TEMLINE = SUBSTR(TEMLINE,VERIFY(TEMLINE,BLANK));                     2 00007540
             IF  (SUBSTR(TEMLINE,1,1) = BRK(3)) THEN                              2 00007550
                       DO;                                                        4 00007550
                       CALL STEXT(LINE);                                          4 00007570
/* ............................................................................ */
                       THFL = TWOB;                                               4 00007580
                       RETURN;                                                    4 00007600
                       END;                                                       4 00007610
/* /* ELSE IS FOLLOWED BY TEXT, SEPARATE  ELSE    */                           */   00007620
             CALL STEXT('ELSE');                                                  2 00007640
/* ............................................................................ */
/* /* REMAINING TEXT, NEW LINE    */                                           */   00007650
             IF SUBSTR(TEMLINE,1,2)=BRK(6)  THEN                                  2 00007651
                       DO;                                                        4 00007651
                       RINDEX=ADDPTR(RINDEX);                                     4 00007652
                       TLINE=LINE;                                                4 00007653
                       L=INDEX(TEMLINE,BRK(7))+1;                                 4 00007654
                       LINE=SUBSTR(TEMLINE,1,L);                                  4 00007655
                       CALL COMMENT;                                              4 00007656
```

```
/* ............................................................................................................ */
                       LINE=TLINE;                                                                               4 00007657
                       TEMLINE=SUBSTR(TEML NE,L+1);                                                              4 00007658
                       TEMLINE=SUBSTR(TEMLINE,VERIFY(TEMLINE,BLANK));                                            4 00007659
                       END;                                                                                      4 00007660
             RINDEX = ADDPTR(RINDEX);                                                                            2 00007661
             COUNT=CCUNT+ONEB;                                                                                   2 00007662
             LINDEX(COUNT)=RINDEX;                                                                               2 00007663
             LINE = TEMLINE;                                                                                     2 00007670
/* /* ANY PREFIX ?   */                                                                                     */     00007680
             CALL RLABEL;                                                                                        2 00007690
/* ............................................................................................................ */
/* /* IS THIS ON IF CR DO STATEMENT   */                                                                    */     00007700
             CMND = SUBSTR(LINE,1,3);                                                                            2 00007710
             IF CMND='BEGIN ' ] CMND='BEGIN:' |  CMND='DO ' | CMND='DO:' THEN                                    2 00007720
                  CLEVEL=CLEVEL-1;                                                                               3 00007720
             END ELSE;                                                                                           2 00007810
FINDCOM:     PROC(LL);                                                                                           2 00007820
/* /* EXAMINES LINE,RETURNS LL = POSTION OF  FIRST COMMA NOT IN QUOTES OR PARENS.  IF NONE, LL = LENGTH (LINE).  */     00007830
/*   */                                                                                                     */     00007830
             DCL (KC,PAREN,LL) BIN FIXED(15,0),                                                                  2 00007860
                  QUOTE  BIT(1) INIT('0'B),                                                                      3 00007860
                  KCHAR CHAR;                                                                                    3 00007860
             PAREN = ZB;                                                                                         2 00007890
L1:               DO KC = 1 TO LENGTH(LINE) - 1;                                                                 3 00007900
                  KCHAR = SUBSTR(LINE,KC,1);                                                                     3 00007910
/* /* DONT LOOK INSIDE QUOTES OR PARENS   */                                                                */     00007920
                  IF (KCHAR = BRK(2)) & (PAREN = ZB) & (¬CUOTE)  THEN                                           3 00007930
                       GO TO FCOM;                                                                               4 00007930
/* ............................................................................................................ */
/* /* IS IT QUOTE OR PAREN ?   */                                                                           */     00007960
                  IF (KCHAR = BRK(1)) THEN                                                                       3 00007970
                            DO;                                                                                  5 00007970
                            QUOTE = ¬QUOTE;                                                                      5 00007990
                            GO TO L3;                                                                            5 00008000
/* ............................................................................................................ */
                            END;                                                                                 5 00008010
                  IF (KCHAR = BRK(4)) THEN                                                                       3 00008020
                       PAREN = PAREN + ONEB;                                                                     4 00008020
                  ELSE                                                                                           3 00008040
                       IF (KCHAR = BRK(5)) THEN                                                                  4 00008040
                                 DO;                                                                             6 00008040
                                 PAREN = PAREN - ONEB;                                                           6 00008070
                                 IF PAREN < ZB THEN                                                              6 00008080
                                      GO TJ PERROR;                                                              7 00008080
/* ............................................................................................................ */
                                 END;                                                                            6 00008100
L3:               END L1;                                                                                        3 00008110
             IF (CUCTE) | (PAREN ¬= 0) THEN                                                                      2 00008120
                  GC TC PERROR;                                                                                  3 00008120
/* ............................................................................................................ */
L4:          LL = LENGTH(LINE);                                                                                  2 00008140
             RETURN;                                                                                             2 00008150
/* /* FOUND COMMA   */                                                                                      */     00008160
FCOM:        LL = KC;                                                                                            2 00008170
             RETURN;                                                                                             2 00008180
PERROR:      PUT SKIP LIST ('**UNBALANCED PARENS OR QUOTES**');                                                  2 00008190
             GO TO L4;                                                                                           2 00008200
/* ............................................................................................................ */
```

```
            END  FINCCOM:                                                              2 00008210
IF:         PROC:                                                                      2 00008220
            DCL  L BIN FIXED(15,0):                                                    2 00008230
/* /* FOR EACH IF CLAUSE   */                                                     */     00008240
            FLAG = '1'B:                                                               2 00008250
OVER:       TYPE(RINDEX) = LTYPE(4):                                                   2 00008260
            SKIP(RINDEX) = ZB:                                                         2 00008270
/* /* ENTER IN PUSHDOWN LIST, NULL LABEL    */                                    */     00008280
            CALL PUSHDON ('IF',MTLAB):                                                 2 00008290
/* ................................................................................ */
            LEVEL(RINDEX)=CLEVEL-1:                                                    2 00008300
/* /* SEPARATE FIRST PHASE THRU THEN    */                                        */     00008310
            L = INDEX(LINE,' THEN '):                                                  2 00008320
            IF L = ZB THEN                                                             2 00008330
                      DO:                                                              4 00008330
                      L = INDEX(LINE,' THEN:'):                                        4 00008350
                      IF L = ZB THEN                                                   4 00008360
                           GO TO THENERR:                                              5 00008360
/* ................................................................................ */
/* /* FOUND 'THEN:', EMPTY CLAUSE    */                                           */     00008380
                      ELSE                                                             4 00008390
                                DO:                                                    6 00008390
                                TLINE=SUBSTR(LINE,1,L+5):                              6 00008410
                                CALL STEXT(TLINE):                                     6 00008411
/* ................................................................................ */
                                GO TO TESTFN:                                          6 00008412
/* ................................................................................ */
                                END:                                                   6 00008430
                      END:                                                             4 00008431
/* /* FOUND 'THEN ' */                                                            */     00008440
            TLINE=SUBSTR(LINE,1,L+4):                                                  2 00008450
            CALL STEXT(TLINE):                                                         2 00008451
/* ................................................................................ */
/* /* UPDATE PTR, LINE, DELETE LEADING BLANKS    */                               */     00008460
            LINE = SUBSTR(LINE,L + 6):                                                 2 00008470
            LINE = SUBSTR(LINE,VERIFY(LINE,BLANK)):                                    2 00008480
            IF SUBSTR(LINE,1,2)=BRK(6) THEN                                            2 00008481
                      DO:                                                              4 00008481
                      RINDEX=ADDPTR(RINDEX):                                           4 00008482
                      TLINE=LINE:                                                      4 00008483
                      L=INDEX(TLINE,BRK(7))+1:                                         4 00008484
                      LINE=SUBSTR(TLINE,1,L):                                          4 00008485
                      CALL COMMENT:                                                    4 00008486
/* ................................................................................ */
                      LINE=SUBSTR(TLINE,L+1):                                          4 00008487
                      LINE=SUBSTR(LINE,VERIFY(LINE,BLANK)):                            4 00008488
                      END:                                                             4 00008489
            RINDEX = ADDPTR(RINDEX):                                                   2 00008490
            COUNT=COUNT+ONEB:                                                          2 00008491
            LINDEX(COUNT)=RINDEX:                                                      2 00008492
/* /* CHECK FOR PREFIX FOR NEW LINE    */                                         */     00008500
            CALL RLABEL:                                                               2 00008510
            IF SUBSTR(LINE,1,3)='DO ' | SUBSTR(LINE,1,3)='DO:' |  SUBSTR(LINE,1,6)='BEGIN ' |  2 00008511
            SUBSTR(LINE,1,6)='BEGIN:'  THEN                                            2 00008511
                 CLEVEL=CLEVEL-1:                                                      3 00008511
/* /* ANOTHER IF ?   */                                                           */     00008520
            IF (SUBSTR(LINE,1,3) = 'IF ') |  (SUBSTR(LINE,1,3) = 'IF(') THEN           2 00008530
                 GO TO OVER:                                                           3 00008530
/* ................................................................................ */
```

```
RET:         THFL = 1;                                                                                               2 00008560
             RETURN;                                                                                                 2 00008570
/* /* NO 'THEN', PRINT MESSAGE, FUDGE, GO ON    */                                                               */   00008580
THENERR:     PUT SKIP LIST('**MISSING ''THEN'' IN IF STATEMENT**');                                                  2 00008590
             PUT SKIP;                                                                                               2 00008600
             CALL STEXT(LINE);                                                                                       2 00008610
/* ............................................................................................................... */
/* /* RETURN TO MAIN PROGRAM    */                                                                               */   00008620
             GO TO P3;                                                                                               2 00008630
/* ............................................................................................................... */
             END IF;                                                                                                 2 00008640
IOSUB:       PROC;                                                                                                   2 00008650
/* /* PROCESSES IO STATEMENTS. SKIP LINE BEFORE  AND AFTER EACH IO OR GROUP OF IO S.    */                       */   00008660
             TYPE(RINDEX) = LTYPE(2);                                                                                2 00008680
             IF (PLEV ¬= 0) THEN                                                                                     2 00008690
                       DO;                                                                                           4 00008690
                       LEVEL(RINDEX) = PLEV;                                                                         4 00008710
                       PLEV = 0;                                                                                     4 00008720
                       END;                                                                                          4 00008730
             ELSE                                                                                                    2 00008740
                  LEVEL(RINDEX) = CLEVEL;                                                                            3 00008740
/* /* IS PRECEDING STATEMENT ALSO IO ?   */                                                                      */   00008760
             IF COUNT = 1 THEN                                                                                       2 00008770
                  GO TO I11;                                                                                         3 00008770
             IF (TYPE(LINDEX(COUNT - 1)) = LTYPE(2)) THEN                                                            2 00008780
                       DO;                                                                                           4 00008780
                       PT = SUBPTR(RINDEX);                                                                          4 00008800
                       IF SKIP(PT) = TWOB THEN                                                                       4 00008810
                            SKIP(PT) = ONEB;                                                                         5 00008810
                       ELSE                                                                                          4 00008830
                            SKIP(PT) = ZB;                                                                           5 00008830
                       SKIP(RINDEX) = 3;                                                                             4 00008850
                       END;                                                                                          4 00008860
             ELSE                                                                                                    2 00008870
I11:              SKIP(RINDEX) = TWOB;                                                                               3 00008870
/* /* ENTER IN BUFFER    */                                                                                      */   00008890
             CALL STEXT(LINE);                                                                                       2 00008900
/* ............................................................................................................... */
             END IOSUB;                                                                                              2 00008910
MOVUP:       PROC(THROO);                                                                                            2 00008920
/* /* CALLED AFTER OUTPUT TO MOVE UP REMAING BUFFER  LINES.  ACTUALLY ONLY THE LINDEX TABLE IS  CHANGED          */   00008930
/* (POINTERS TO FIRST LINE OF EACH STATEMENT  IN BUFFER(. COUNT IS THE NUMBER OF STATEMENTS  IN BUFFER, THROO    */   00008930
/* IS THE NUMBER OF LAST ST  TO BE MOVED OUT. UPDATES COUNT,RINDEX    */                                         */   00008930
             DCL (THROO,JCT) BIN FIXED(15,0);                                                                        2 00008990
             COUNT = COUNT - THROO;                                                                                  2 00009000
             IF COUNT = ZB THEN                                                                                      2 00009010
                  RETURN;                                                                                            3 00009010
                  DO JCT = 1 TO COUNT;                                                                               3 00009030
                  LINDEX(JCT) = LINDEX(JCT + THROO);                                                                 3 00009040
                  END;                                                                                               3 00009050
/* /* ZERO UNUSED INDICES    */                                                                                  */   00009060
                  DO JCT = COUNT + 1 TO 20;                                                                          3 00009070
                  LINDEX(JCT) = ZB;                                                                                  3 00009080
                  END;                                                                                               3 00009090
             END MOVUP;                                                                                              2 00009100
ONSUB:       PROC;                                                                                                   2 00009110
/* /* PROCESSES ON CONDITION, SIGNAL AND REVERT  STATEMENTS. PREDEDES  STATEMENTS. PRECEDES EACH BY BLANK SPACE  */   00009120
/* AND  DOTTED LINE, FOLLOWS BY DOTTED LINE, UNLESS  ON ST. INCLUDES A BEGIN GROUP, WHEN  ONFLAG IS SET TO 1     */   00009120
/* AND THE FOLLOWING DOTTED  LINE IS IMPLEMENTED AFTER THE CORRESPONDING  END.    */                             */   00009120
```

```
             DCL    LL FIXED BIN(15,0);                                                                 2 00009200
             TYPE(RINDEX) = LTYPE(6);                                                                   2 00009210
             IF (PLEV ¬= 0) THEN                                                                        2 00009220
                       DO;                                                                              4 00009220
                       LEVEL(RINDEX) = PLEV;                                                            4 00009220
                       PLEV = 0;                                                                        4 00009240
                       END;                                                                             4 00009250
             ELSE                                                                                       2 00009260
                  LEVEL(RINDEX) = CLEVEL;                                                               3 00009260
             SKIP(RINDEX) = 7;                                                                          2 00009280
/* /* IS LAST (NON-BLANK) WORK 'BEGIN' ?    */                                                      */    00009290
             IF LENGTH(LINE) < 7 THEN                                                                   2 00009300
                  GO TO TSTOR;                                                                          3 00009300
/* .............................................................................................................. */
L1:               DO LL = LENGTH(LINE) - 1 TO 6 BY -1;                                                  3 00009320
                  IF SUBSTR(LINE,LL,1) = BLANK THEN                                                     3 00009330
                       GO TO L2;                                                                        4 00009330
                  IF SUBSTR(LINE,LL-5,6) = ' BEGIN' THEN                                                3 00009350
                            DO;                                                                         5 00009350
                            ONFLAG = '1'B;                                                              5 00009370
                            SKIP(RINDEX) = 6;                                                           5 00009380
                            END;                                                                        5 00009390
                  GO TO TSTOR;                                                                          3 00009400
/* .............................................................................................................. */
L2:               END    L1;                                                                            3 00009410
TSTOR:       CALL STEXT(LINE);                                                                          2 00009420
/* .............................................................................................................. */
             END ONSUB;                                                                                 2 00009430
OUTPUT:      PROC(THRU);                                                                                2 00009440
/* /* PRINTS STATEMENTS 1 - THRU    */                                                              */    00009450
/* /* WRITTEN FOR 80 OR 120 COLUMN PRINTOUT  ON SYSTEM FILE    */                                   */    00009460
             DCL (RI,THRU, NV, LASTN, I) BIN FIXED(15,0),                                               2 00009480
                  (LEV, MC) DEC FIXED (3,0),                                                            3 00009480
                  (CMTB INIT ('/* '), CMTE INIT (' */')) CHAR(3);                                       3 00009480
             IF THRU = ZB THEN                                                                          2 00009510
                  RETURN;                                                                               3 00009510
/* /* ALL LINES ARE AT SAME LEVEL (RULE 1)    */                                                    */    00009530
             LEV = LEVEL(LINDEX(1));                                                                    2 00009540
             MC = MARGIN(LEV) + 1;                                                                      2 00009550
/* /* FIND POINTER TO LAST OUTPUT LINE    */                                                        */    00009560
             IF THRU = COUNT THEN                                                                       2 00009570
                  LASTN = RINDEX;                                                                       3 00009570
             ELSE                                                                                       2 00009590
                  LASTN = SUBPTR(LINDEX(THRU + 1));                                                     3 00009590
             RI = SUBPTR(LINDEX(1));                                                                    2 00009610
/* /* PRINT ONE LINE AT A TIME    */                                                                */    00009620
LPO:              DO I = 1 TO 50;                                                                       3 00009630
                  RI = ADDPTR(RI);                                                                      3 00009640
                  LEV=LEVEL(RI);                                                                        3 00009641
                  MC=MARGIN(LEV)+1;                                                                     3 00009642
/* /* IS IT COMMENT ?    */                                                                         */    00009650
                  IF TYPE(RI) = LTYPE(7) THEN                                                           3 00009660
                            DO;                                                                         5 00009660
OCMT:                       PUT FILE(SYSPRINT) EDIT  (CMTB,TEXT(RI),CMTE,SEQ#(RI))                      5 00009680
                            (COL(2),A,A,COL(NCOL-2),A,X(4),A);                                          5 00009680
                            GO TO LPA;                                                                  5 00009710
/* .............................................................................................................. */
/* /* GB */                                                                                     */    00009711
                            END;                                                                        5 00009711
```

```
/* /* NOT COMMENT: IS TEXT NULL ?   */                                                                          */      00009720
                  IF TEXT(RI) = '' THEN                                                                               3 00009730
                            DO;                                                                                       5 00009730
NTXT:                       PUT FILE(SYSPRINT) EDIT(PREFIX(RI,*),  LABEL(RI,*),LEV,SEQ#(RI))  (COL(2),10              5 00009750
                            A,COL(NCOL+2),F(2),X(1),A);                                                               5 00009750
                            GO TO LPA;                                                                                5 00009780
/* ............................................................................................................ */
                            END;                                                                                      5 00009790
/* /* LINE HAS TEXT    */                                                                                       */      00009800
TXT:              PUT FILE(SYSPRINT) EDIT  (PREFIX(RI,*), LABEL(RI,*),  TEXT(RI),LEV,SEQ#(RI))  (COL(2), (10)A,       3 00009810
                  COL(MC), A, COL(NCOL+2),  F(2),X(1),A);                                                             3 00009810
LPA:              IF RI = LASTN THEN                                                                                  3 00009860
                       GO TO LPOUT;                                                                                   4 00009860
/* ............................................................................................................ */
LPE:              END LPO;                                                                                            3 00009880
/* /* ALL LINES PRINTED, LOOK FOR AFTER SHIP OR DOT    */                                                       */      00009890
/* /* BY RULES 2 & 3, ONLY LAST LINE CAN HAVE FOLLOWING SKIP OR  DOT    */                                      */      00009900
LPOUT:       IF SKIP(LASTN) = 0 THEN                                                                                  2 00009920
                  RETURN;                                                                                             3 00009920
             ELSE                                                                                                     2 00009940
                  NV = SKIP(LASTN);                                                                                   3 00009940
/* /* PRINT DOT, SKIP    */                                                                                     */      00009960
PUNC:        IF (NV = 5) | (NV = 4) THEN                                                                              2 00009970
                  IF (NCOL = 72) THEN                                                                                 3 00009970
                       PUT FILE(SYSPRINT) EDIT  (CMTB, (65) '.', CMTE)  (COL(2), A, A, COL(70), A);                   4 00009970
                  ELSE                                                                                                3 00010020
                       PUT FILE(SYSPRINT) EDIT  (CMTB, (112) '.', CMTE)  (COL(2), A, A, COL(117), A);                 4 00010020
             IF (NV = 5) | (NV = 3) THEN                                                                              2 00010060
                  PUT SKIP;                                                                                           3 00010060
             RETURN;                                                                                                  2 00010080
/* /* IF VERY FIRST STATEMENT HAS PRE-SKIP  ETC., ENTER HERE    */                                               */      00010090
OUTFST:      ENTRY(NV2);                                                                                              2 00010110
             IF (NV2 = 1) THEN                                                                                        2 00010120
                  NV = 3;                                                                                             3 00010120
             ELSE                                                                                                     2 00010140
                  IF (NV2 = 6) THEN                                                                                   3 00010140
                       NV = 5;                                                                                        4 00010140
                  ELSE                                                                                                3 00010170
                       IF (NV2 = 8) THEN                                                                              4 00010170
                            NV = 4;                                                                                   5 00010170
             GO TO PUNC;                                                                                              2 00010200
/* ............................................................................................................ */
             END OUTPUT;                                                                                              2 00010210
PEND:        PROC;                                                                                                    2 00010220
/* /* PROCESSES END ST, WHICH MAY TERMINATE  PROC, DO, OR BEGIN BLOCK, UPDATES  PUSHDOWN LIST AND LEVEL    */   */      00010230
             DCL LAB CHAR (31) VAR;                                                                                   2 00010260
             DCL LVAR LABEL   (FAN3, PLAINL);                                                                         2 00010270
             DCL (LIND, IFAN) BIN FIXED(15,0);                                                                        2 00010280
             TYPE(RINDEX) = LTYPE(4);                                                                                 2 00010290
             LEVEL(RINDEX) = CLEVEL;                                                                                  2 00010300
/* /* SKIP IF THIS IS END OF A CONDITION BLOCK    */                                                            */      00010310
/* /* BEGINNING GB */                                                                                           */      00010320
             IF ONFLAG THEN                                                                                           2 00010321
                  SKIP(RINDEX)=4;                                                                                     3 00010321
             ELSE                                                                                                     2 00010322
                  SKIP(RINDEX)=ZB;                                                                                    3 00010322
/* /* STORE TEXT  */                                                                                            */      00010323
             CALL STEXT(LINE);                                                                                        2 00010324
             IF (ONFLAG) THEN                                                                                         2 00010325
```

```
              DO:                                                     4 00010325
              ONFLAG='0'B:                                            4 00010326
              RETURN:                                                 4 00010327
              END:                                                    4 00010328
/* /* END OF GB */                                                  */    00010329
/* /* IS END FOLLOWED BY IDENTIFIER ?   */                          */    00010410
        TEMLINE = SUBSTR(LINE,4):                                     2 00010420
        TEMLINE = SUBSTR(TEMLINE,VERIFY(TEMLINE,BLANK)):              2 00010430
        IF LENGTH(TEMLINE) = 1 THEN                                   2 00010440
           GO TO PLAIN:                                               3 00010440
/* ..................................................................*/
/* /* SET IDENTIFIER INTO LAB   */                                  */    00010460
/* /* GB */                                                         */    00010470
FANCY:  LAB=SUBSTR(TEMLINE,1,LENGTH(TEMLINE)-1)||':':                 2 00010471
        LIND = INDEX(LAB,BLANK):                                      2 00010480
        IF LIND ¬= ZB THEN                                            2 00010490
/* /*  GB  */                                                       */    00010490
           LAB=SUBSTR(LAB,1,LIND-1)||':':                             3 00010490
/* /* TEST FOR ERROR IN PUSHDOWN LIST   */                          */    00010510
FAN2:   LVAR = FAN3:                                                  2 00010520
        GO TO STERR:                                                  2 00010530
/* ..................................................................*/
/* /* RETURN HERE IF NO ERROR   */                                  */    00010540
/* /* DOES IDENTIFIER MATCH STACK LABEL ?   */                      */    00010550
FAN3:      DO IFAN = 1 TO 5:                                          3 00010560
           IF (STACK2(IFAN) = LAB) THEN                               3 00010570
              IF (STACK1='ENTRY') THEN                                4 00010570
                 GO TO ENTERR:                                        5 00010570
              ELSE                                                    4 00010600
                 GO TO HAVIT:                                         5 00010600
/* ..................................................................*/
/* /* IDENTIFIER NOT IN THIS STACK LEVEL   */                       */    00010620
           END:                                                       3 00010630
        IF STACK1 = 'ENTRY' THEN                                      2 00010640
           FLAG = '0'B:                                               3 00010640
        ELSE                                                          2 00010660
           FLAG = '1'B:                                               3 00010660
        CALL POPUP(STACK1):                                           2 00010680
/* ..................................................................*/
/* /* KEEP LOOKING FOR LAB   */                                     */    00010690
        GO TO FAN2:                                                   2 00010700
/* ..................................................................*/
/* /* JUST END NO IDENTIFIER   */                                   */    00010710
/* /* TEST FOR STACK ERROR   */                                     */    00010720
PLAIN:  LVAR = PLAIN1:                                                2 00010730
        GO TO STERR:                                                  2 00010740
/* ..................................................................*/
/* /* RETURN HERE IF OK   */                                        */    00010750
PLAIN1: IF STACK1 = 'ENTRY' THEN                                      2 00010760
              DO:                                                     4 00010760
              FLAG = '0'B:                                            4 00010780
              CALL POPUP(STACK1):                                     4 00010790
/* ..................................................................*/
              GO TO PLAIN:                                            4 00010800
/* ..................................................................*/
              END:                                                    4 00010810
/* /* POPUP 1 LEVEL   */                                            */    00010820
HAVIT:  FLAG = '1'B:                                                  2 00010830
        CALL POPUP(STACK1):                                           2 00010840
```

```
             IF STACK1='IF' THEN                                                                                      2 00010850
                  CLEVEL=STACK3;                                                                                      3 00010850
             CLEVEL=STACK3;                                                                                           2 00010851
             LEVEL(RINDEX)=CLEVEL+1;                                                                                  2 00010852
             RETURN;                                                                                                  2 00010870
/* /* ERROR RETURSN*/                                                                                           */      00010880
ENTERR:      PUT SKIP LIST('**LABEL ON END STATEMENT SHOULD NOT MATCH '||  'ENTRY NAME**');                           2 00010890
/* /* TRY TO RECOVER*/                                                                                          */      00010910
             GO TO PLAIN1;                                                                                            2 00010920
/* ............................................................................................................. */
/* /* TEST FOR STACK ERROR */                                                                                   */      00010930
STERR:       IF STACK1='TOP' | CLEVEL=0 THEN                                                                          2 00010940
                       DO;                                                                                            4 00010940
                       PUT SKIP LIST('**UNMATCHED END OR ERROR IN IF...ELSE '||  'STRUCTURE**');                      4 00010960
                       STOP;                                                                                          4 00010980
                       END;                                                                                           4 00010990
             GO TO LVAR;                                                                                              2 00011000
/* ............................................................................................................. */
             END PEND;                                                                                                2 00011010
POPIF:       PROC;                                                                                                    2 00011020
/* /* POPIF (CALLED BY PUSHPUL) CR ENTRY PUPEL  (CALLED BY ELSE OR PEND) CLEARS PUSHDOWN  LSIT WHEN IF, IF ...   */      00011030
/* ELSE PAIR, OR BLOCK  IS TERMINT  IS TERMINATED. CALLS POPUP TO DO CLEARING.   */                              */      00011030
             DCL  ELFLAG BIT(1) INIT('0'B);                                                                           2 00011080
/* /* POPUP TOP 'IF'   */                                                                                       */      00011090
POP1:        FLAG='0'B;                                                                                               2 00011100
             CALL POPUP('IF');                                                                                        2 00011110
/* ............................................................................................................. */
/* /* IS LIST EMPTY ?   */                                                                                      */      00011120
             END POPIF;                                                                                               2 00011370
POPUP:       PROC(CMAND);                                                                                             2 00011380
/* /* POPS UP 1 STACK LEVEL (IN EACH OF THE  3 LISTS), DECREMENTS CLEVEL IF FLAG = 1   */                        */      00011390
             DCL CMAND CHAR(*) VARYING;                                                                               2 00011410
/* /* NO LEVEL CHANGE IF ENTRY OR ELSE   */                                                                     */      00011420
             IF (FLAG) THEN                                                                                           2 00011430
                  CLEVEL = CLEVEL - 1;                                                                                3 00011430
             IF (STACK1 ¬= CMAND) THEN                                                                                2 00011450
                       DO;                                                                                            4 00011450
                       PUT SKIP LIST ('**ERROR IN STACK1**');                                                         4 00011470
                       STOP;                                                                                          4 00011480
                       END;                                                                                           4 00011490
/* /* POPUP 1 LEVEL IN EACH STACK   */                                                                          */      00011500
             FREE STACK1,STACK2,STACK3;                                                                               2 00011510
             END POPUP;                                                                                               2 00011610
PUSHDON:     PROC(CMAND,LABL);                                                                                        2 00011620
/* /* ENTERS OP. NAME (E.G. 'IF') IN  STACK1, UP TO 5 LABELS IN  STACK2, CLEVEL IN STACK3.  STORAGE FOR THE      */      00011630
/* STACKS IS CONTROLLED.  ALLOCATED IN PUSHDON, FREED IN POPUP   */                                             */      00011630
             DCL CMAND CHAR(*) VAR,                                                                                   2 00011680
                  LABL(5) CHAR(*) VAR,                                                                                3 00011680
                  X DEC FIXED(2);                                                                                     3 00011680
/* /* GET PRESENT LEVEL   */                                                                                    */      00011710
             X = STACK3;                                                                                              2 00011720
/* /*  INCREMENT LEVEL UNLESS ENTRY, ELSE, OR IF */                                                             */      00011730
             IF (FLAG) THEN                                                                                           2 00011740
                  X = X + 1;                                                                                          3 00011740
/* /* PUSHDOWN EACH OF 3 STACKS   */                                                                            */      00011760
             ALLOCATE STACK1,                                                                                         2 00011770
                  STACK2(5),                                                                                          3 00011770
                  STACK3;                                                                                             3 00011770
             STACK3 = X;                                                                                              2 00011780
```

```
                 STACK1 = CMAND;                                                                          2 00011790
                 STACK2 = LABL;                                                                           2 00011800
    /* /* UPDATE CURRENT LEVEL    */                                                               */       00011810
                 CLEVEL = STACK3;                                                                         2 00011820
                 END PUSHDON;                                                                             2 00011830
    PUSHPUL:     PROC;                                                                                    2 00011840
    /* /* CALLED BY PLEDIT OR RETURN FROM READ TO CHECK ON  PRESENT STATUS OF IF...ELSE STRUCTURES AND DO ANY  */   00011850
    /* NECESSARY UPDATING OF THE PUSHDOWN STOCKS    */                                             */       00011850
                 DCL  L BIN FIXED(15,0);                                                                  2 00011880
                 THFL = 0;                                                                                2 00011890
    /* /* ELSE STATEMENT ?    */                                                                   */       00011900
                 IF (SUBSTR(LINE,1,5) = 'ELSE ') | (SUBSTR  (LINE,1,5) = 'ELSE;') THEN                    2 00011910
                        DO;                                                                               4 00011910
                        CALL ELSE;                                                                        4 00011940
                        IF THFL=TLOB THEN                                                                 4 00011941
                           GO TO P3;                                                                      5 00011941
    /* ..................................................................................................... */
                        RETURN;                                                                           4 00011950
                        END;                                                                              4 00011960
    /* /* NO; IS TOP OF STACK 'IF' ?    */                                                         */       00011970
    AGIN1:       IF STACK1='ELSE' THEN                                                                    2 00011980
                    CALL POPUP('ELSE');                                                                   3 00011980
                 IF STACK1='IF' THEN                                                                      2 00011981
                        DO;                                                                               4 00011981
                           DO WHILE(STACK1='IF');                                                         5 00011982
                           CLEVEL=STACK3-1;                                                               5 00011983
                           CALL POPIF;                                                                    5 00011984
    /* ..................................................................................................... */
                           END;                                                                           5 00011985
                        GO TO AGIN1;                                                                      4 00011986
    /* ..................................................................................................... */
                        END;                                                                              4 00011987
                 RETURN;                                                                                  2 00011990
    /* /* YES, IF ST. IS COMPLETED, CHECK OFF   */                                                 */       00012000
                 END PUSHPUL;                                                                             2 00012020
    SMARG:       PROC;                                                                                    2 00012030
    /* /* GIVEN READ-IN VALUES OF IMARGIN (INITIAL  MARGIN) AND DELMARG (MARGIN INCREMENT) THIS  SETS UP A TABLE OF  */   00012040
    /* MARGIN VALUES FOR NESTING  LEVELS 1 - 9. COMMENT STATEMENTS ARE LEVEL 0  AND  0 MARGIN IS DEFINED AS 1   */   */   00012040
                 DCL IMA FIXED BIN(15,0);                                                                 2 00012090
                 MARGIN(1) = IMARGIN;                                                                     2 00012100
                    DO IMA=2 TO 15;                                                                       3 00012110
                    MARGIN (IMA) = MARGIN (IMA - 1) + DELMARG;                                            3 00012120
                    END;                                                                                  3 00012130
                 END SMARG;                                                                               2 00012140
    STEXT:       PROC (SOMLIN);                                                                           2 00012150
    /* /* WRITTEN FOR 80 OR 120 COLUMN PRINTOUT (NCOL = 72  OR 116). STORES STATEMENT TEXT IN BUFFER IN  PRINT LINE  */   00012160
    /* QUANTA, ALLOWING FOR PREFIXES AND  LABELS. DIVIDES TEXT AT WORD ENDS IF POSSIBLE  SEPARATES FORE-AND-AFT   */   00012160
    /* PARTS OF SKIPCODE AND  STORES THEM APPROPRIATELY.  */                                       */       00012160
                 DCL (CFL,COMFL) BIT(1) INIT('0'B);                                                       2 00012220
                 DCL (LP,LL,CC) BIN FIXED(15,0) INIT(Z8);                                                 2 00012230
                 DCL (NCHAR, MG, RC, NI, FC, WSKIP) FIXED BIN(15,0),                                      2 00012240
                    TLEV DEC FIXED(2),                                                                    3 00012240
                    SOMLIN CHAR(*) VARYING;                                                               3 00012240
                 SEQ#(RINDEX)=SEQNO;                                                                      2 00012261
                 TLEV = LEVEL(RINDEX);                                                                    2 00012270
                 MG = MARGIN(TLEV);                                                                       2 00012280
    /* /* SEPARATE SKIPCODE INTO FOR/AFT COMPONENTS    */                                          */       00012290
                 WSKIP = SKIP(RINDEX);                                                                    2 00012300
                 FC = CHCODE(WSKIP,1);                                                                    2 00012310
```

```
             RC = CHCCDE(WSKIP,2);                                                2 00012320
/* /* COMMENT HAS DIFF'T PUNCT., NO PREFIX   */                                 */  00012330
             IF TYPE(RINDEX) = LTYPE(7) THEN                                      2 00012340
                       DO;                                                        4 00012340
                       NCHAR = NCOL - 7;                                          4 00012360
                       COMFL = '1'B;                                              4 00012370
                       GO TO AGAIN;                                               4 00012380
/* .............................................................................. */
                       END;                                                       4 00012390
/* /* NOT COMMENT   */                                                          */  00012400
             NCHAR = NCOL - MG;                                                   2 00012410
                  DO NI = 1 TO 5;                                                 3 00012420
                  LP = LP + LENGTH(PREFIX(RINDEX,NI));                            3 00012430
                  LL = LL + LENGTH(LABEL(RINDEX,NI));                             3 00012440
                  END;                                                            3 00012450
/* /* DO PREFIX, LABEL NEED SEPARATE LINE(S) ?   */                             */  00012460
             IF LP + LL ¬> MG THEN                                                2 00012470
                  GO TO AGAIN;                                                    3 00012470
/* .............................................................................. */
/* /* YES, TOO LONG   */                                                        */  00012490
             IF LP = ZB THEN                                                      2 00012500
                  GO TO A2;                                                       3 00012500
/* .............................................................................. */
/* /* PREFIX PRESENT: GIVE IT A LINE   */                                       */  00012520
             CC = CC + ONEB;                                                      2 00012530
             SKIP(RINDEX) = FC;                                                   2 00012540
             PT = ADDPTR(RINDEX);                                                 2 00012550
             LABEL(RINDEX,*)='';                                                  2 00012560
             TEXT(RINDEX)='';                                                     2 00012561
             RINDEX = PT;                                                         2 00012570
             SEQ#(RINDEX)=SEQNO;                                                  2 00012571
/* /* NOW LABEL, IF ANY   */                                                    */  00012580
/* /* SHORT ENOUGH TO FIT IN MARGIN ?   */                                      */  00012590
             IF LL  <= MG THEN                                                    2 00012600
                       DO;                                                        4 00012600
                       CFL = '1'B;                                                4 00012620
                       GO TO AGAIN;                                               4 00012630
/* .............................................................................. */
                       END;                                                       4 00012640
/* /* LONGER, SEPARATE   */                                                     */  00012650
A2:          CC = CC + ONEB;                                                      2 00012660
             IF CC = CNEB THEN                                                    2 00012670
                  SKIP(RINDEX) = FC;                                              3 00012670
             ELSE                                                                 2 00012690
                       DO;                                                        4 00012690
                       SKIP(RINDEX) = 0;                                          4 00012710
                       PREFIX(RINDEX,*)='';                                       4 00012720
                       TYPE(RINDEX)='';                                           4 00012721
                       END;                                                       4 00012730
             TEXT(RINDEX)='';                                                     2 00012740
             LEVEL(RINDEX) = TLEV;                                                2 00012750
             RINDEX = ADDPTR(RINDEX);                                             2 00012760
             SEQ#(RINDEX)=SEQNO;                                                  2 00012761
/* /* SEPARATE TEXT INTO PRINT LINES, STORE   */                                */  00012770
AGAIN:       CC = CC + ONEB;                                                      2 00012780
/* /* CASE: PR + LABEL < MARGIN   */                                            */  00012790
             IF CC = CNEB THEN                                                    2 00012800
                  SKIP(RINDEX) = FC;                                              3 00012800
/* /* BUT GENERALLY:   */                                                       */  00012820
```

```
            ELSE                                                                                          2 00012830
                      DO;                                                                                 4 00012830
                      SKIP(RINDEX) = 0;                                                                   4 00012850
                      IF (CFL) THEN                                                                       4 00012860
                           CFL = ¬CFL;                                                                    5 00012860
                      ELSE                                                                                4 00012880
                           LABEL(RINDEX,*) = '';                                                          5 00012880
                      PREFIX(RINDEX,*) = '';                                                              4 00012900
                      IF (¬COMFL) THEN                                                                    4 00012910
                           TYPE(RINDEX) = '';                                                             5 00012910
                      ELSE                                                                                4 00012930
                           TYPE(RINDEX) = LTYPE(7);                                                       5 00012930
                      END;                                                                                4 00012950
            LEVEL(RINDEX) = TLEV;                                                                         2 00012960
/* /* WILL TEXT FIT IN THIS LINE ?   */                                                                       */    00012970
            IF LENGTH(SOMLIN) <= NCHAR THEN                                                               2 00012980
                      DO;                                                                                 4 00012980
                      TEXT(RINDEX) = SOMLIN;                                                              4 00013000
                      GO TO TEXTOUT;                                                                      4 00013010
/* ...........................................................................................................*/
                      END;                                                                                4 00013020
/* /* NO, SEPARATE BETWEEN WORDS   */                                                                         */    00013030
                 DO NI = NCHAR TO 1 BY -1;                                                                3 00013040
                 IF (SUBSTR(SOMLIN,NI,1) = BLANK) THEN                                                    3 00013050
                      GO TO LINOUT;                                                                       4 00013050
/* ...........................................................................................................*/
                 END;                                                                                     3 00013070
/* /* NO BLANK FOUND   */                                                                                     */    00013080
            NI = NCHAR;                                                                                   2 00013090
LINOUT:     TEXT(RINDEX) = SUBSTR(SOMLIN,1,NI);                                                           2 00013100
            SOMLIN = SUBSTR(SOMLIN,NI + 1);                                                               2 00013110
            IF SOMLIN = '' THEN                                                                           2 00013120
                 GO TO TEXTOUT;                                                                           3 00013120
/* ...........................................................................................................*/
            RINDEX = ADDPTR(RINDEX);                                                                      2 00013140
            SEQ#(RINDEX)=SEQNO;                                                                           2 00013141
            GO TO AGAIN;                                                                                  2 00013150
/* ...........................................................................................................*/
/* /* TEXT COMPLETE, CORRECT SKIP FOR 'AFTER'    */                                                           */    00013160
TEXTOUT:    IF CC = CNEB THEN                                                                             2 00013170
                 SKIP(RINDEX) = WSKIP;                                                                    3 00013170
            ELSE                                                                                          2 00013190
                 SKIP(RINDEX) = RC;                                                                       3 00013190
/* /* RETURN   */                                                                                             */    00013210
            END STEXT;                                                                                    2 00013220
/* /* ALL SUBROUTINES AND FUNCTICNS HAVE BEEN INCLUDED   */                                                   */    00013230
        END PLEDIT;                                                                                       1 00013240
```